

CULLER/HARRISON, INC.
150-A Aero Camino
Goleta, California 93017
(805) 968-1064

September 1978

AD-A155 461

STRUCTURED DESIGN

FOR AN

LPC ARRAY PROCESSOR

Quarterly Technical Report

Research on Speech Compression Prototype Development

May 1978 - August 1978

PROJECT DIRECTOR: Dr. Glen J. Culler

DTIC
ELECTE
JUN 17 1985
S D
G

This research was sponsored by the
Defense Advanced Research Projects
Agency under ARPA Order No.: 3625
Contract No.: MDA903-78-C-0313
Expiration Date: March 20, 1979
Monitored by: DCASMA Oxnard, Calif.

Distribution of this document is
unlimited. It may be released
to the Clearinghouse, Department
of Commerce for sale to the
general public.

APPROVED FOR PUBLIC RELEASE,
DISTRIBUTION IS UNLIMITED (A)

The views and conclusions contained in this document are those of the authors
and should not be interpreted as necessarily representing the official policies,
either expressed or implied, of the Advanced Research Projects Agency or the
U.S. Government.

85 06 13 141

DTIC FILE COPY

CONTENTS

	<u>Page</u>
1. INTRODUCTION	1
A. Summary of Contract Intent	1
B. Current Stage of Development	2
2. PRELIMINARY DESIGN FOR AN LPC ARRAY PROCESSOR	5
A. The APU Module	9
B. The CPU Module	11
C. The PS Module	14
D. The I/O Module	14
3. THE STRUCTURED DESIGN TECHNIQUE	16
A. Reconveyance of Design	17
B. Computer Involvement in the Design Process	18
C. Mathematical Model for Computer Design	19
D. Defining Logical Modules	26
E. The Integrated Logical Simulator	32
F. The Integrated Module Analyzer	37

APPENDIX A: Sample Outputs from the Structured Design System

APPENDIX B: Micro Programs

APPENDIX C: Module Analyzer Logical Drawings



Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A/	

STRUCTURED DESIGN FOR AN LPC ARRAY PROCESSOR

1. INTRODUCTION

The research activities of the several ARPA contractors in the NSC network speech compression program have reached a stage of success in LPC algorithm development to merit real consideration of hardware economy and efficiency. To be sure, there are many other contributors to the body of research supporting the LPC technique; and most all of it taken together forms a successful whole that needs to be supported by better hardware to facilitate its expression. To this end, ARPA has funded a modest effort at CHI to design an array processor that is particularly well suited to perform the important steps of the family of LPC algorithms and at the same time to be amenable to LSI implementations. What follows is our first quarterly report on this design effort.

A. Summary of Contract Intent

The design team at CHI has been engaged in developing programmable special purpose computers for a number of years. Much of this work has been done for ARPA, principally in the direction of on-line research systems for signal processing in the acoustical frequency domain.

Our past work has a certain architectural stamp or style that results from the way we have approached the design process, the blending of very different technical backgrounds, and the techniques we have we settled on for solving internal structure problems that seem to have resulted in surprising efficiency and reliability of our devices. In our past efforts we have generally been confronted with an applicational design goal and with a fixed technology* in terms of which the design must be realized. With the rapid change of technology, we have found ourselves designing the same internal structures in different ways to take advantage of the new technology. This has led us to ask how may we represent our designs so they can be independent of the technology used to realize them. The task of designing an LPC array

*In this instance, what we mean by a technology is a set of available digital circuits and associated connectors and construction techniques.

processor that can be prototyped from existing components and then later realized as a single (or a few) LSI chips, brings this technology invariance question forward as a primary issue to be dealt with.

As a result of this, the intent of our work on contract MDA903-78-C-0313 is not only to interpret the basic operations occurring in various LPC analysis and synthesis algorithms and to design an array processor that efficiently performs them, but also to carry out the research and development required to produce a design system that will make it possible to produce technology-invariant designs. We are not suggesting that this can be done "untouched by human hands," for this is not in the nature of architecture; but rather that the human input to the design system will be primarily limited to providing creative organizational expression while the computer part of the system responds to this input by generating a number of important design outputs such as a logic simulator that realizes an equivalent design, and after geometrical specification and primitive module specification, a connector sort, a logic sort, and ultimately, diagnostic software. Of course, all of this points in the direction of how to use a computer system to help us establish how successful a design is before we build it and how to prove it works correctly after it's built.

B. Current Stage of Development

By the time contract work began we had come to understand in qualitative terms the two principal issues involved:

1. What do the LPC equations indicate about the choice of array processor architecture and how does this relate to earlier array processors.
2. What are the important software objects that could be expected to work together to achieve a structured design system that would help us solve the reconveyance problem.

Consequently, we were able to allocate tasks that could proceed in parallel. These turned out to be of two kinds -- those that were clearly understood and would play a definite roll in the work to be accomplished and those that would gather information that might help us find out how to deal with the intangibles of our problem. In the latter category we had to:

- a. Compare the equations arising in representative LPC algorithms and establish performance criteria at the array process level.
- b. Review the steps of our prior design efforts and the associated documentation and seek a restructuring of both these steps and the documentation to make the whole of it computationally derivable from a structured design document.
- c. Study potential data structures and associated control functions that could lead to a logical simulator that is computationally derivable from a structured design document.

At present, we can report on successful accomplishment of these tasks and are well along in implementing their consequences. More specifically:

1. We have a design for an LPC array processor ready for testing.
2. A user input language for specifying a logical design has been devised and a program has been written -- called the logic assembler -- that deduces a data structure which represents the logical design and, as such, provides the basis for logic simulation.
3. We have established the concept of module substitution within the logic simulator and have designed a module analyzer and its interface to the ARPA signal processing system. By specifying to the logic assembler that a certain module is to be replaced during simulation by an actual hardware module, we can cause the logic simulator to control the module analyzer and capture the hardware module's outputs for further use in the simulation. What this means is that with the device represented by software, we can test a given hardware module "in situ" for the whole device without having the rest of the device in hand.
4. We have restructured the nature of our hardware documentation to achieve a form that can be automatically generated and yet be of similar use to engineers in both checkout and maintenance. Indeed, an engineer has to become familiar with its use, but its simplicity appears to make this no great difficulty.

5. Programs that produce design outputs of this selected documentation format are being written and some intermediate results are ready for use.
6. We have begun specifying the test programs for the LPC array processor.

The work remaining to be done on the contract consists of building and checking out the prototype, finishing the software for the structured design system and demonstrating the performance of the prototype on representative LPC algorithms.

In discussions leading up to this contract work it was anticipated that after the array processor design was appropriately in hand, we would be in a position to cooperate with some selected LSI house as an aid in the technology transfer. Our estimate is that by the end of the second quarter we will be in a position to do this.

2. A PRELIMINARY DESIGN FOR AN LPC ARRAY PROCESSOR

In J. Makhoul's paper "Stable and Efficient Lattice Methods for Linear Prediction," the best known methods of LPC analysis are brought together in a simple common framework. We choose this as our principal reference, partly because of its clarity, but mostly because it reveals a wide range of computational complexities engaged in the LPC techniques. Boiling these down to their essences results in only a few prime considerations, nevertheless arranging for their adequate implementation brings us as close to a general purpose array processor as we dare risk. The point of concern is this -- when a family of tasks are to be carried out in a single special purpose processor, they must possess some underlying simplicity that results in design advantage or we are forced to a general purpose design and the power of our approach is lost. Under the latter circumstances we must restrict the family before we can achieve a successful design.

The input to the analysis process comes from an A/D converter which may as well be taken in integer format. The dynamic range of voice signals is such that at least 8 bits/sample are required to perform any sensible analysis and 10 bits are as low as most implementers are willing to consider. In fact, most systems of reasonable quality either use 12-bit samples or incorporate some form of secondary gain ranging. In what follows we will assume that the input data is 12 bits/sample and in integer format. As to conversion rates, the NSC conferencing system used 150 μ s/sample corresponding to 3.2 KHz Nyquist frequency which is marginally acceptable for male voices but unfortunately low for some female users with very small vocal tracts. A sample interval of 50 μ s guarantees good sibilant data for practically all users, so we assume that the sample interval ΔT is in the range $50 \mu\text{s} \leq \Delta T \leq 150 \mu\text{s}$. In order to set up an array oriented process, we pick a data block size long enough to get statistical benefit in estimating the vocal tract area function and short enough to get good resolution of the change of this shape with time. Accordingly, we can expect analysis block sizes to be in the neighborhood of 100 samples with corresponding time bases in the range of 5 to 15 milliseconds.

In keeping with Makhoul's notation, let f and b represent forward and backward residuals and K denote reflection coefficient. Since $-1 < K < 1$, the direct lattice filter equations:

$$\begin{aligned} f_{m+1}(n) &= f_m(n) + K_{m+1}b_m(n-1) \\ b_{m+1}(n) &= K_{m+1}f_m(n) + b_m(n-1) \end{aligned} \quad 1)$$

requires us to perform:

$$(\text{Fraction} \times \text{Integer})_{\text{RND}} + \text{Integer} \rightarrow \text{Integer} \quad 2)$$

as an array process. The inverse filter:

$$\begin{aligned} f_m(n) &= f_{m+1}(n) - K_{m+1}b_m(n-1) \\ b_{m+1}(n) &= K_{m+1}f_m(n) + b_m(n-1) \end{aligned} \quad 3)$$

incorporates feedback and for small amplitudes engenders a limit cycle pathology. To handle these properly, we propose to incorporate an integer by fraction multiplication with rounding and to overlap the add (or subtract) operation. Additionally, to guard against overflow, we will arrange that only 11 of the 12 bits of integer data will be significant going into the above equations. Also, as a means to stay away from the limit cycle pathology, we will sense the output size information on the fly. That is, when the input data is too large, we will use a built-in right shift in the array process and when the output data is too small, we will insert an auxiliary left shift array process.

- a) If the 12th bit of the input arrays is significant, right shift on read.
- b) If the output array has no more than 8 significant bits, left shift it 3 places before proceeding.

In the computation of inner products of integer arrays like:

$$f \cdot f, b_{-1} \cdot b_{-1}, f \cdot b_{-1} \quad \text{where } b_{-1}(n) = b(n-1)$$

we need to use integer-by-integer multiplication, and since our arrays have about 100 elements, we need an integer multiply overlapped with addition into an extended accumulator.

The noncorrective methods require much more precision than the corrective ones; for this purpose, we provide programmed floating point and double precision operations. We are all aware of the much lower operation count of these methods compared to the reduction of residual corrective methods, but to be fair, we should count them in at least the double precision sense. In many cases, not having the residuals as an output of the computations creates a need for extra processes in the overall algorithm, thereby further reducing their apparent advantage.

In this section we present our current results in the development of an array processor tailored to the requirements of real time LPC analysis and synthesis. In keeping with the contract intent discussed in section 1, we will later express the design in such a manner that it can be equivalently realized in different technologies. To do this we will use the structured design approach presented in section 3 with our overall design goals established as a creative interpretation of the requirements set forth through the review of typical LPC algorithms.

The highest level top-down subdivision of the LPCAP splits it into four submodules: APU, CPU, PS, and IO.

Their overall functions can be described as follows:

1. The APU module carries out all the arithmetic operations on data as specified by the right-hand part of the instruction lines (INSTRR). Its registers provide source data for output to the IO module. Certain characteristic bits provide status information to the CPU that may be used in CPU conditionals.
2. The CPU module performs program sequencing, data addressing, indexing, APU and IO control. Its operations are specified by the left-hand part of the instruction lines (INSTRL). In run mode these instruction lines are supplied by some of the outputs of PS. When a HOST is present and when the CPU is halted, these instruction lines are supplied by the IO module, thus providing the HOST with single step control of the CPU.

3. The PS module is a program source module addressed by the CPU. When PS is a RAM (or contains a subset that is a RAM) it can be loaded by a sequence of outputs from the IO module under control of the HOST.
4. The IO module provides an interface between the LPCAP and the external world. Aside from whatever complexities are introduced by the presence of a HOST machine, it must manage voice A/D and D/A conversion and receive and transmit voice parameters to some communication link.

The main substance of this section is the treatment of each of these modules in turn.

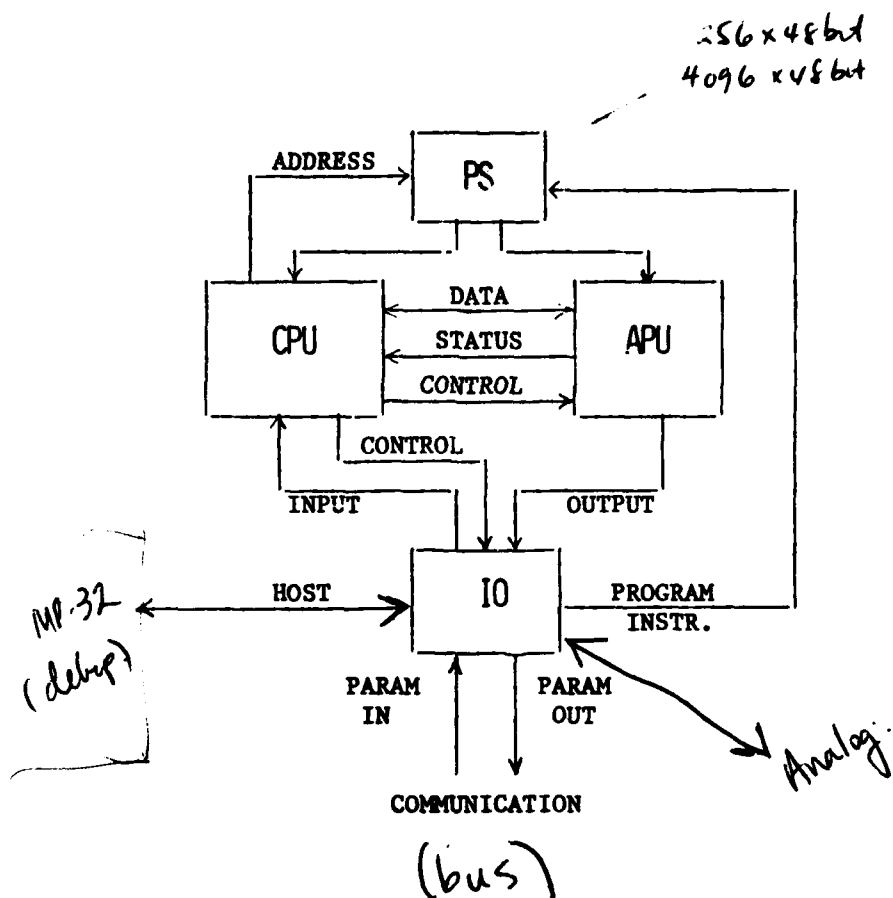


Figure 2.1 Block diagram showing the relationship of the four principal submodules of the LPCAP.

A. The APU Module

The arithmetic requirements on the APU module imposed by the form of the equations used in LPC analysis and synthesis may be summarized as follows:

1. The dynamic range of voice signals are best represented by 12-bit samples.
2. Precise inner products of sample vectors with at least 128 components are required for the noncorrective methods of computing reflection coefficients.
3. Fraction by integer multiplication is required for forming linear combinations of signals.
4. Both down scaling to guard against overflow and upscaling to guard against limit cycle pathologies of digital filtering are required.
5. The same inputs must apply to both sides of the multiplier so that the sum of squares can be computed without copying a vector.
6. The quotients count in LPC algorithms is so low that a programmed divide will suffice.

Since the LPC algorithms have a very concentrated requirement for multiplies and since most of these are associated with adds, we should focus our attention on multipliers that have configurations with both integer-by-integer full precision and fraction-by-integer rounded outputs. Such multipliers are already available with execution times in the neighborhood of microprocessor instruction times so the architectural considerations in loading and unloading data is of paramount importance. A powerful illustration of how to do this right and the unfortunate consequences of not doing it right is provided by comparing multiply-add throughputs of the TRW chips MP12AJ and MP16AJ. The first can be used with no loss time in a looped multiply-add and the latter has a loss essentially matching its execution time. Our point is not to flamboyantly snipe at the MP16AJ but rather to emphasize the value of the architecture of the MP12AJ. Even though LSI technology will soon support the complete LPCAP on one chip, the internal multiply submodule should have the architecture of the MP12AJ.

With the architecture of the multiply nailed down, we can extend the structure to account for the requirements listed above.

- a. Since the forward and backward waves are the primary inputs to the multiplier in a Burg-type analysis, and since the sum of squares of each is to be computed, we must provide a 2-to-1 multiplexer for each of the multiplier inputs (cf figure 2.2); then 5) is satisfied.
- b. In order to provide precise inner products of 12-bit data in 128 word blocks, we use the full integer product with sign extension to 32 bits and use this as the A inputs to a 32-bit adder, with a 32-bit accumulator attached to the B side; this takes care of 2).
- c. Since the reflection coefficients are fractions between -1 and +1, when they multiply a vector with integer components to give a vector of the same type we must use the fraction-by-integer product with rounding. To add this product to another such vector, as in the lattice filter computation, we also need a 12-bit accumulator.
- d. The scaling problems intrinsic to LPC analysis and synthesis are not very severe, yet we must have some means for handling them. One simple way to do this is to maintain two flag bits under program control, call them Big and Small. Let Big selectively monitor the data written in pad and express the or of the exclusive ors of the left-hand two bits of each array value. Then Big is a 1 if any array datum is of full scale. Now let Small monitor the left-hand 4 bits in the same manner. Then Small is a 0 if no array datum has more than 8 significant bits. The capability of clearing and testing these flag bits and maintaining a scale word in pad to account for change of scale, provides a software overview of the scaling relations. We include downscaling hardware on the output from pad and will use the multiply hardware in a special pass to upscale the data when required.
- e. One of the aspects of the special forms of the equations relating the forward and backward waves coupled with the direct association of adders and accumulators is that we sometimes need to exchange outputs. Perhaps a stronger way to say this is that we must be able to move data between scratch pad arrays. To satisfy this desire, we include a pair of 2-to-1 output multiplexers.

- f. The final step in this overview of the APU consists of providing means for assembling and disassembling byte-oriented data sets. To this end we have selected two-way-in registers for the accumulators with interconnections to an 8-bit accumulator accepting an external input.

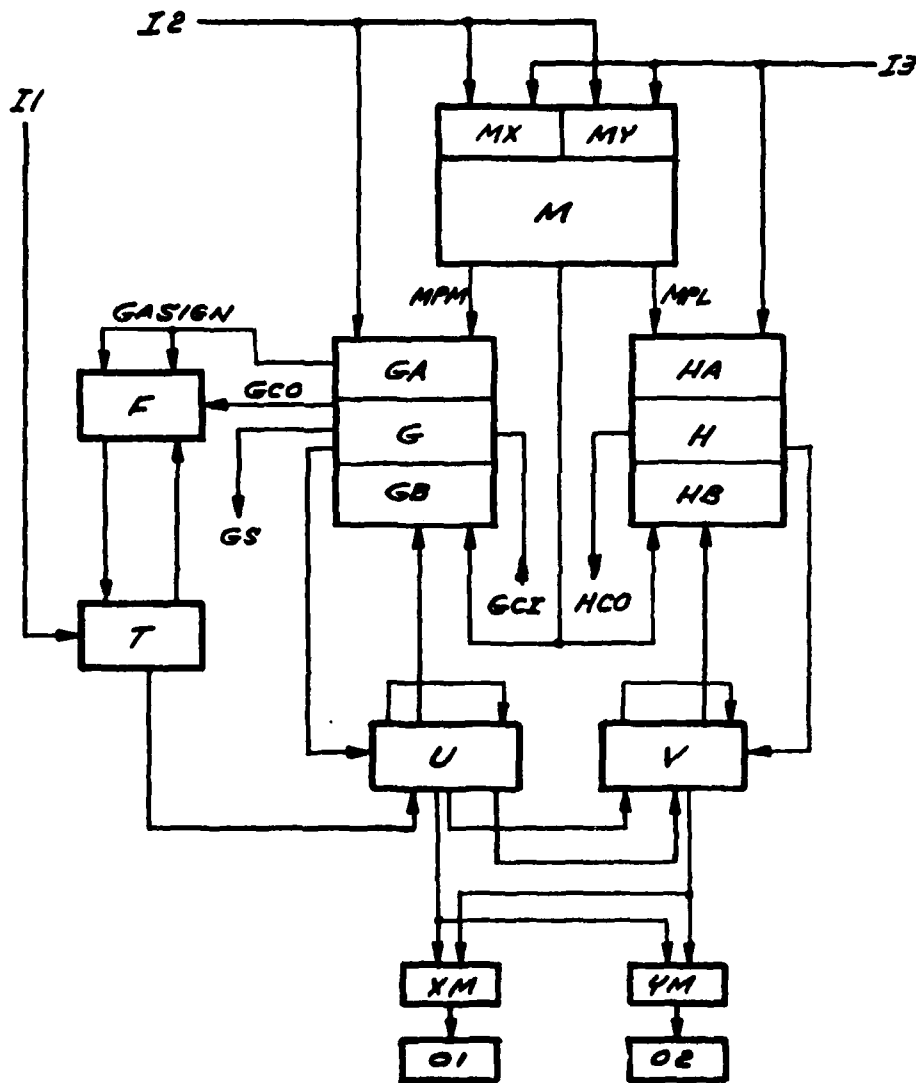
In figure 2.2 below, we give a module schematic for the APU together with a decoding chart with the module operation code definitions.

B. The CPU Module

The APU module discussed above carries with it all of the significance of design relating to LPC specializations. For the CPU we are after the simplest sufficiently rapid control to get the job done. Whereas a 12-bit structure was used in the APU module, an 8-bit structure is the proper choice for the CPU module. The two scratch pads (X and Y) of the APU need only be 256 words long to provide adequate data buffering, so corresponding to these the CPU contains two 8-bit address registers (XA and YA). Present estimated program size indicates that with an instruction page size of 256 instructions, we can expect the subroutines for analysis and synthesis to each fit within a page. For prototyping purposes we will use a 4096 instruction program source (the PS module discussed in C) with 48-bit instructions and 256 per page. In a real device this will, of course, be limited to the size determined by programming the algorithms.

The CPU can now be summarized as follows. Starting with three address counters (PSA, XA, YA) of 8 bits and an 8-bit tally register for array loop control we bus these together with 8 bits from program source (the PS value field). Branching according to the tally register, the sign bit of the APU adder and IO status bits are then provided.

In setting up an array process, the bus connection between the PS value field and the internal registers of the CPU is used to initialize them one at a time. In the array process, all of the registers can be simultaneously controlled in each micro instruction. In order to provide control for the IO module, the bus mentioned above is extended to the IO module. In the bus control fields (TRN and REC) shown in figure 2.4 below, we can address a device and then input or output as required.



$21 + 27 = 48$
bits

	4	4	1	1	5
	U	V	G	H	M
	UEN	VEN	GA	HA	MULT
CODE 0	NOOP	NOOP	0	0	RND
1	NOOP	NOOP	SUBT	SUBT	MXCLK
2	G+U	H+V	RSUBT	RSUBT	MYCLK
3	RSTU	RSTV	ADD	ADD	MXSEL
4			EOR	EOR	MYSEL
5			OR	OR	
6			AND	AND	
7			-1	-1	

Figure 2.2 APU Module Schematic

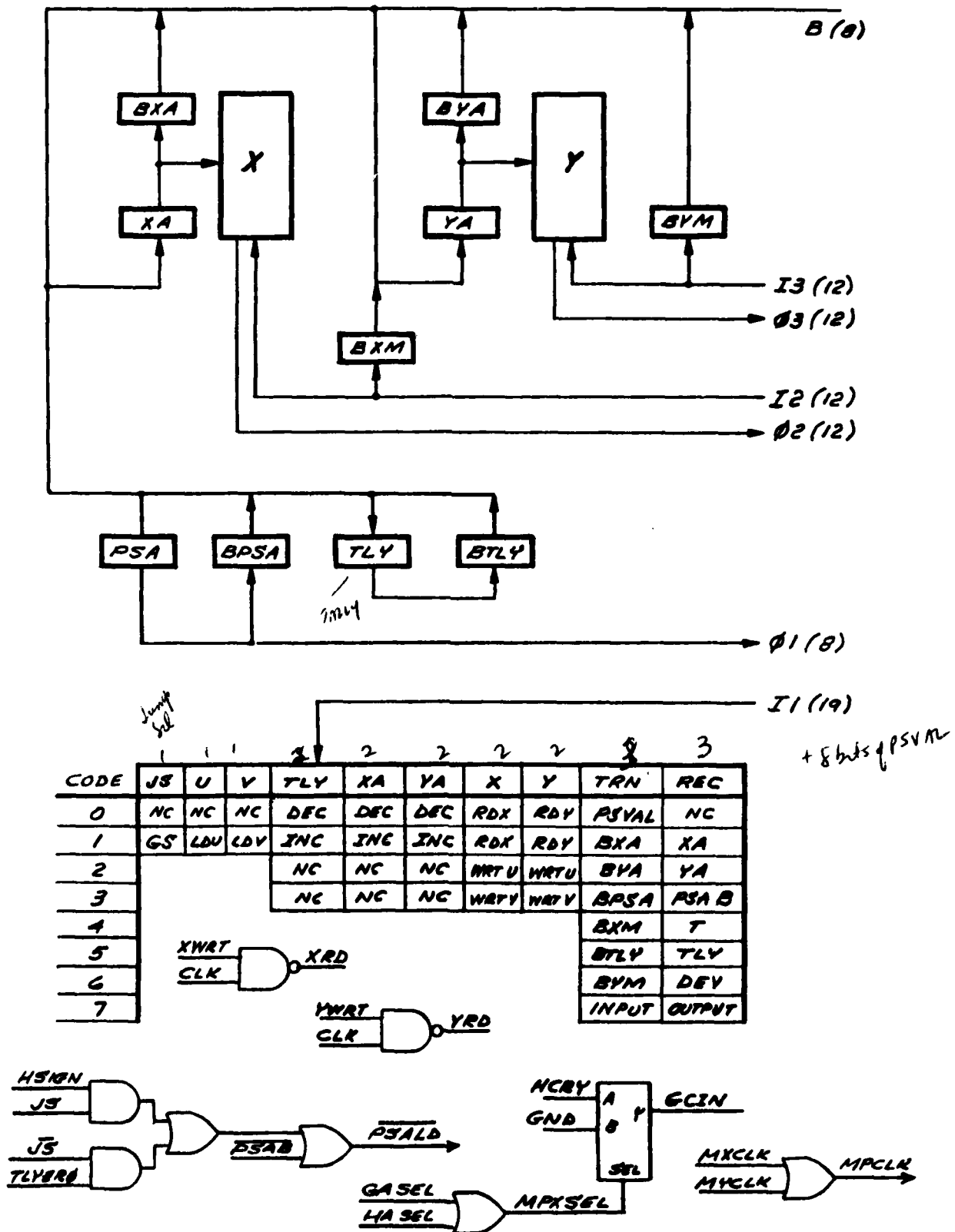


Figure 2.3 CPU Module Schematic

C. The PS Module

The program source module has been designed to allow full convenience in implementing trial LPC algorithms that will need to assess the real time performance of the LPC AP. The design is elementary and merits no special discussion.

D. The IO Module

The design of the IO module has been specialized to provide an interface between the LPC AP and the Serial Three signal processing research system. Our approach here has been to take advantage of the module analyzer -- one of the key elements in the structured design system discussed in section 3. The module analyzer is directly interfaced to the host machine and can be used as a programmable test fixture to check out the individual modules of the LPC AP as well as the entire device.

The IO module provides a means for loading programs into PS, transferring sample data blocks and parameter blocks to and from the Host. Additionally, it provides the logic of a pseudo panel that permits the Host to single step operations of the LPC AP.

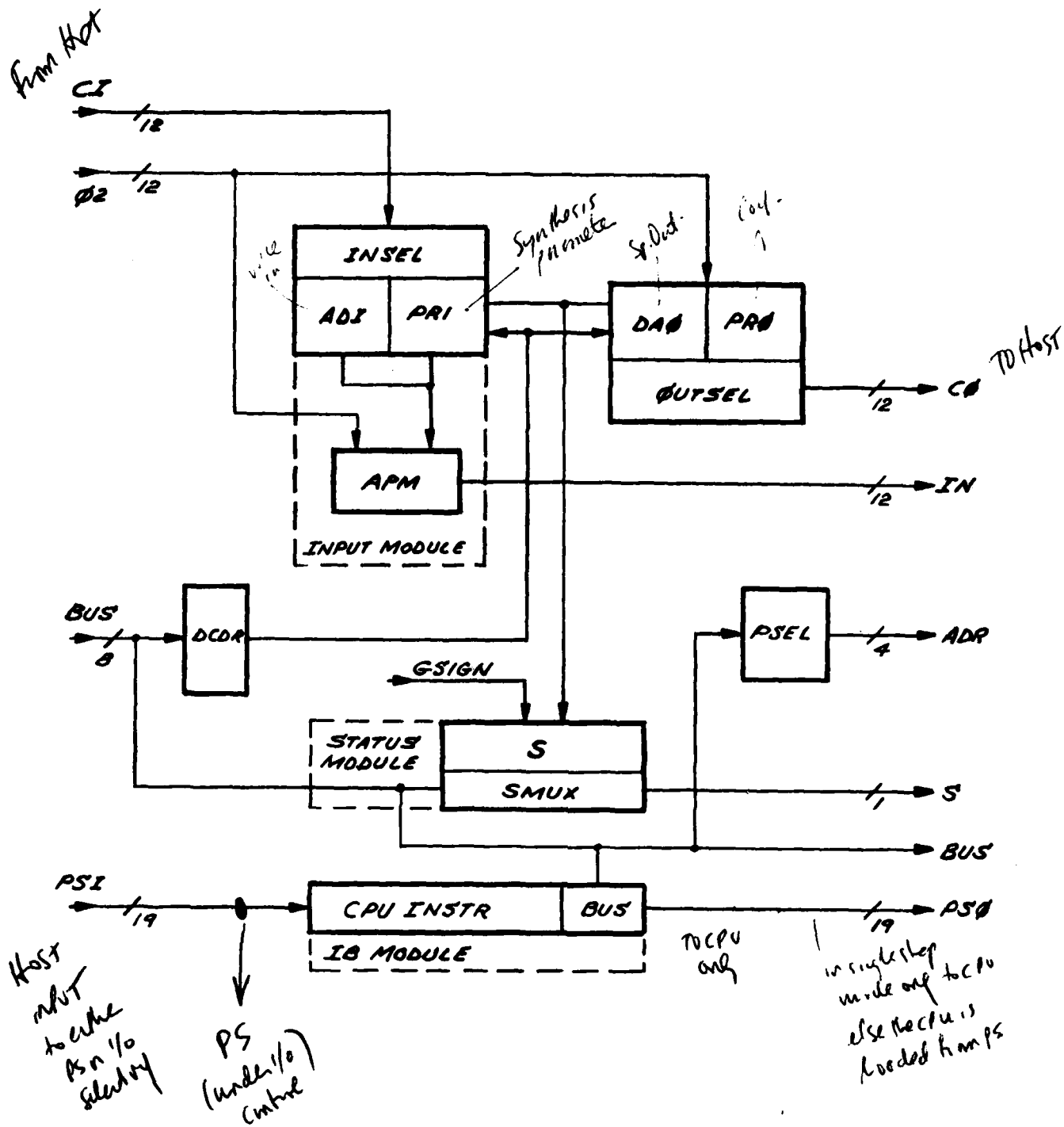


Figure 2.4 IO Module Schematic

3. THE STRUCTURED DESIGN TECHNIQUE

Logical design can be thought of as a creative process in which larger modules are defined as combinations of smaller, simpler ones in such a way as to fulfill basic design goals.

The approach can be "bottom up." In this case the designer visualizes combinations of available parts, and combinations of these, always working towards accomplishing the desired goals.

An alternative is a "top down" approach in which the design is subdivided into sections which, if implementable, would achieve the basic goals. These sections are again decomposed, and so on, until the resulting blocks are seen to be buildable from available parts. The subdivisions represent intermediate design targets.

A third approach works from both directions simultaneously, hoping to meet in the middle. At that point, sections which are known to accomplish the design goals are seen to be realizable as combinations of combinations of available hardware.

In any of these approaches the final design can be conceived as a single entity or expressed in modular terms. However, modular design simplifies checkout, allows for consideration of more design alternatives, and facilitates making changes.

A structured design is a tree-like hierarchy having the following characteristics.

1. For each module in the structure, the outputs are uniquely determined by the inputs, the controls and the status of memory.
2. Each module in the hierarchy is either primitive (not broken down further) or composite (a combination of lower level modules called the submodules of the module).
3. The definition of a composite module involves only its submodules. No reference to submodules of the submodules is allowed.

This means that from the point of view of a module, there is only one level below it. All strata beneath are invisible to the module, and each module acts as a primitive relative to higher level design.

In a hardware realization of the design, the primitives might be chips, and lower level, intermediate modules might be groups of chips wired together. A higher level intermediate module might be realized as a board or a group of boards connected by cables.

A. Reconveyance of Design

Assume that a design has been created which accomplishes the original goals. Of course, many alternative designs could be produced fulfilling the same criteria. In particular, one of the intermediate modules (along with its substructure) could be replaced by a different combination of submodules (along with their substructures) which performed the same function. If all of the relationships among other elements of the hierarchy were preserved, the resultant design would fulfill the original goals. Similarly, substitution for an intermediate module in the new design would lead to an additional acceptable design.

Thus, one design can be reconveyed by a whole family of equivalent designs. The highest level subdivision of the original design is of critical importance since all members of the family inherit the top structure.

If an intermediate module has been thoroughly checked out, either by theoretical simulation or by prototyping and hardware checkout, the design can be reconveyed into one in which the module in question is a primitive. The design is simplified by removing the substructure of the module from the hierarchy.

This is of practical as well as theoretical interest. For, at any stage, from design through prototyping or even production, technological advances can be easily incorporated into the existing design. If a function which was previously computed in a composite module is found to be performable by a new device (for example, a chip) the original design can be reconveyed into one where that module is a primitive. It would not be necessary to redesign the whole system, which would be required if there were no modularity.

B. Computer Involvement in the Design Process

A computer can be brought into the design process in several ways.

Elements of a computer aided design system are given below.

Given an inventory of primitives and a language suitable for describing composite modules, a logic assembler can produce a data structure encapsulating the design. From this structure, appropriate programs can generate the tedious documentation necessary to implement the design -- for example, connector sort and wire list. This frees the designer to concentrate on more creative aspects such as alternative designs of intermediate targets.

An editor which enables changes to be made in documents defining the design facilitates consideration of alternatives.

A logical simulator can be used to calculate the response of any module in the system to particular sets of inputs and controls (not, of course, in real time). The simulator works on the data structure set up by the logic assembler. At each stage, the inputs and controls are passed on to submodules until they reach a primitive module. Then a subroutine pointed to in the primitive's description is invoked. The outputs thus calculated are passed back to the higher level module. Eventually, when the actions of all submodules of a module have been simulated, the outputs of that module are implicitly determined. In this way, the design can be exercised, while still in theoretical form, to make sure that each module conforms to design goals.

When a module has been realized by engineering design and prototyping, its adequacy as part of the overall design can be tested by integrating the action of a module analyzer with the simulator. The hardware realization of the module is plugged into the analyzer and switches are set to indicate which connector elements represent controls inputs and outputs.

At the stage in the simulation when execution of the module is required, inputs and controls are supplied to the external device through the module analyzer. The resulting outputs produced by the device become the values which are used at the next stage of the simulation.

C. A Mathematical Model for Computer Design

In order to implement a computer aided, design facility it is necessary to have an underlying mathematical model of a computer system and the process by which the system is constructed from components. This 'theoretical' computer, as well as its components, will be called a logical module. The model should be such that its features and functions can be translated into a computer acceptable language.

The purpose of this section is to provide basic definitions and clarify the synthesis process by revealing rules of combination which result in logical modules. We must also develop a language that provides precise expression of one logical module in terms of others. These concerns motivate the definitions which follow.

The definitions are in terms of vector spaces, so as a preliminary step, we indicate notations which will be used elsewhere in this section.

Let X and Y be finite dimensional, vector spaces over finite fields.

1. $X \times Y$, the cross-product of X and Y , is the set of pairs of vectors (x, y) where $x \in X$ and $y \in Y$.
2. If X and Y are over the same field, then $X \oplus Y$, the direct sum of X and Y , is the set of vectors $x|y$. If

$$x = (x_1, \dots, x_n) \in X \text{ and } y = (y_1, \dots, y_m) \in Y$$

then $x|y$ has coordinate expression

$$x|y = (x_1, \dots, x_n, y_1, \dots, y_m)$$

3. If X and Y are of the same dimension, n , and over the same field, they are isomorphic and can be identified. This identification will be denoted $X \leftrightarrow Y$.
4. The dimension of X will be denoted by $\dim X$

Logical Modules

Defn 1: A Logical Module L is an ordered sextuple

$$L = \langle C, I, IO, M, O, F \rangle$$

where C, I, IO, M, O are finite dimensional vector spaces over finite fields and F is a function from $C \times I \times IO \times M$ into O such that, for each set of vectors $c \in C, i \in I, io \in IO, m \in M$ there is a unique vector $o \in O$ for which

$$F(c, i, io, m) = o$$

The notation used here has the following mnemonic relations:

C is the control space of the module

I is the input space of the module

IO is the input/output space of the module

M is the memory space of the module

O is the output space of the module

F is the function of the module

IO and/or M may be empty. The vectors of IO represent either input or output depending upon the control vector.

The outputs can be 'switchable,' in which case the field associated with O has at least three elements.

For simplicity O is presented as a single space; however, if some but not all outputs are switchable, O is really a pair of vector spaces.

The crux of the definition dictates that for every combination of control, input, IO and memory vectors, there is a unique output vector 'computed' by the function F .

The concept of a logical module applies over a broad range of complexities, extending from 'chips' to an entire computer system.

Defn 2: Let L be a logical module. The boundary of L , denoted by δL , is the quadruple

$$\delta L = \langle C, I, IO, O \rangle$$

The quantity

$$\dim \delta L = \dim C + \dim I + \dim IO + \dim O$$

will be called the dimension of the boundary.

The boundary is not itself a vector space, since the underlying fields of the constituent spaces may be different. However, the individual coordinates of the constituent spaces can be ordered to run from 1 to $\dim \delta L$ (as for $x|y$). This will be called an ordering of the elements of the boundary.

Defn 3: A logical module L is primitive if its function F is explicitly given as part of the module's definition.

Defn 4: A logical module L is composite if it is defined by a combination of logical modules

$$L_1, L_1, L_2, \dots, L_n$$

The L_i are called submodules of the module L . A submodule can be either primitive or itself a composite module. Only primitive modules can have memory.

In order to combine logical modules and form more comprehensive structures which are themselves logical modules, we must specify a boundary correspondence that defines a boundary δL for a composite module L in terms of the boundaries δL_i of the constituent submodules. The boundary elements of δL must be associated with boundary elements of submodules. A single element of δL may be associated with elements in more than one submodule or even with more than one boundary element of a single module. A further restriction on the boundary correspondence is that output elements of the composite module boundary be associated with output elements of the submodules. (It is not required that, for example, each input to the composite module be an input to some submodule.)

It will usually be the case that there are some submodule boundary elements which are not associated with elements of the composite boundary. Mappings may be specified among such elements.

These mappings are called interconnects.

From the definitions above, it seems clear that the function of a composite module is derivable from the structural relationships of the boundaries of its submodules and the submodule functions. In fact, the boundary correspondences and interconnects define the function F of the composite module L implicitly in terms of the functions F_i of the submodules L_i .

To aid us in giving some specific examples of combinations, we introduce the following notions.

Defn 5: Two logical modules are equivalent if there exists a one-one correspondence of their boundaries which results in their functions becoming identical.

Defn 6: A composite logical module is homogeneous if all of its submodules are equivalent.

Simple Composite Modules

As a rule it is difficult to describe the functional relationships between the function F of the composite module L and the functions F_i of the submodules. In such a case, the boundary correspondences and interconnects can be listed element by element. However, when the boundary mappings can be described in vector terms, F often has a simple representation in terms of the F_i . We detail a few such situations below.

Parallel Modules

In the design of a computer system, we frequently encounter logical modules that occur in parallel configuration; that is, their inputs, outputs, memories and controls are respectively independent and consequently their functions can be simultaneously used.

For example, consider an 'or' chip with 4 'or' gates which can be used independently. In this case the individual gates, not the chip, are the primitives; the chip represents a composite module with parallel submodules.

Let

$$L_1 = \langle C_1, I_1, IO_1, M_1, O_1, F_1 \rangle \quad \text{and}$$

$$L_2 = \langle C_2, I_2, IO_2, M_2, O_2, F_2 \rangle$$

be logical modules; let

$$C = C_1 + C_2 \quad I = I_1 + I_2 \quad IO = IO_1 + IO_2$$

$$M = M_1 + M_2 \quad O = O_1 + O_2 \quad \text{and define}$$

$$F(c, i, io, m) = F_1(c_1, i_1, io_1, m_1) | F_2(c_2, i_2, io_2, m_2).$$

for each combination of control input IO and memory vectors.

Then $L = \langle C, I, IO, M, O, F \rangle$ is a logical module with parallel submodules. The boundary of L is $\langle C_1 + C_2, I_1 + I_2, IO_1 + IO_2, O_1 + O_2 \rangle$. $\dim \delta L = \dim \delta L_1 + \dim \delta L_2$. There are no interconnects.

Widened Modules

Using homogeneous submodules it is easy to define a wider module with the same controls. Since homogeneous modules are logically equivalent, we may use the ordering assigned by the equivalence and identify corresponding controls. This, in effect, gangs the controls of the submodules together. The inputs, outputs and memories are combined in parallel as above.

Let

$$C = C_1 \leftrightarrow C_2 \quad I = I_1 + I_2 \quad IO = IO_1 + IO_2$$

$$M = M_1 + M_2 \quad O = O_1 + O_2 \quad \text{and define}$$

$$F(c, i, io, m) = F(c, i_1, io_1, m_1) | F_2(c, i_2, io_2, m_2)$$

for each combination of control, input, IO and memory vectors.

$$\dim \delta L = \dim C_1 + 2 \dim I_1 + 2 \dim IO_1 + 2 \dim O_1 < \dim \delta L_1 + \dim \delta L_2$$

There are no interconnects.

An example of the realization of a widened module is a 12-bit register made from three 4-bit register chips.

Composition of Modules

Logical modules may be combined by composition in two important ways, viz the output of one may provide inputs or controls for another. In either case, the composition mapping represents interconnects.

Function Composition

If the dimension of the output space of the first module agrees with the dimension of the input space of the second module, then the composition of these is a logical module. Let

$$C = C_1 + C_2 \quad I = I_1 \quad IO = IO_1 + IO_2$$

$$M = M_1 + M_2 \quad O = O_2 \quad \text{and define}$$

$$F(c, i, io, m) = F_2(c_2, F_1(c_1, i_1, io_1, m_1), io_2, m_2)$$

for each combination of control, input, IO, and memory vectors.

Such modules are, in effect, cascaded and, in the case that L_1 can be preparing its new outputs without modifying its prior outputs, this composition is said to form a pipeline.

Control Composition

If the output space of one module coincides with the control space of another module, the composition of these modules is a logical module.

Let $\dim O_1 = \dim C_2$,

$$C = C_1 \quad I = I_1 + I_2 \quad IO = IO_1 + IO_2$$

$$M = M_1 + M_2 \quad O = O_2 \quad \text{and define}$$

$$F(c, i, io, m) = F_2(F_1(c_1, i_1, io_1, m_1), i_2, io_2, m_2)$$

for each combination of control, input, IO and memory vectors.

In such a case as this we say the module L_1 has provided the control for module L_2 . Such a module is called a control module.

In the various composite modules treated above, we have limited ourselves to simple combinations, yet combinations of these combinations will provide most of what is required to treat significant computer systems.

D. Defining Logical Modules

The definitions of the previous sections were of conceptual intent; that is, they essentially assigned formal names to intuitive concepts. In order to translate these concepts into somethings a computer can get its teeth into, we must present a module definition language and a corresponding data structure. Since module combinations can be completely described in terms of the module boundary and the boundaries of the submodules, the language is a vehicle for specifications of boundaries.

Three kinds of documents suffice for logical module definitions:

- 1) definition document for a primitive module
- 2) definitive document for a composite module
- 3) definition document for a module which is a copy of a previously defined module (either primitive or composite).

Definition document for a primitive module

The boundary elements of the module must be ordered and assigned 'local' names. For example, if the module is to be realized as a 'chip,' the names given to the pins by the manufacturer will suffice. The document then consists of a list of those names, together with a description of the use made of each boundary element (control, input, IO, output). The amount of memory required (if any) must be detailed.

In addition, pointers must be provided to subroutines which compute the module's function. Up to three such subroutines are allowed. This allows control, input, IO and output vectors to be divided into convenient subsets, and enables subsets of outputs to be computed from subsets of inputs.

Definition document for a composite module

The document contains a list of submodules of the module. Then, for each element of the composite module boundary, two items of information must be given: the use of this element (input, control, etc.) and the signal name associated with that element. Each signal name must appear in the definition document of at least one submodule.

The signal names in the document definitions for the module and its submodules determine the boundary correspondences and the interconnects between the submodules. The major design step, then, is encompassed in the assignment of signal names.

Definition document for a copy of a previously defined module

In such a document, the name given in the original definition document must be provided. Then the boundary elements of the module are assigned new signal names. The elements are listed in the same order as in the original module definition document. For each boundary element, its type (control, input, etc.) as well as its assigned (new) signal name is designated.

Using such documents allows many copies to be made from one definition document. In the case of primitives, this avoids proliferation of copies of the subroutine for the module. In the case of composite modules, it enhances modularity by allowing one set of signal names to be associated with boundary elements within a module, and a different set to be assigned to those same elements when the module is used as a component of a larger module.

If the definition document for a submodule L_1 of a module L is itself a composite module, definition document, then the signal names connecting boundary elements of L with boundary elements of L_1 will continue into the interior of L_1 . Such a signal connects at least three levels of the structure. This runs counter to modularity but is allowed in our system for flexibility.

When the design documents for all submodules of a module are copies of previously defined modules, all signals among the boundary elements of the module and the submodules stop at the boundaries of the submodules, preserving modularity. That is, wiring between modules is independent of wiring within those modules.

Examples of definition documents

We illustrate the concepts of the previous sections by including printouts of design documents.

Document 1 is a composite module, definition document for an 8-bit counter, CNTR8. It is composed of the submodules CNT1, CNT2, CNTB, and has 24 boundary elements. Signal names on the same line represent elements of the same type -- control, input, etc. The signal names listed here appear in the design definition documents for the modules CNT1, CNT2 and CNT8.

Document 2 defines a module ADDR8 which is a copy of the CNTR8 module. ADDR8 is combined with another module to form a module PAD. The definition document for PAD is shown in document 3.

Those signal names in document 2 which also appear in document 3, (e.g. BUS 7) detail part of the boundary correspondence for the composite module PAD. Names which appear in document 2 but not in document 3 (e.g. A7) will occur in the definition document for the other submodule of PAD (RAM12). These represent interconnects between the submodules of PAD.

```

DOCUMENT CNTR8          PAGE      1    LINE      1
*CNTR8 8 bit counter module with bussed and regular
output
SUBMODS      CNT1,CNT2,CNTB
CONTROL      LDn,INC,BA,BAn,CLK,G1n,G2n
IO           BUS7,BUS6,BUS5,BUS4,BUS3,BUS2,BUS1,BUS0
OUTPUT       CNT7,CNT6,CNT5,CNT4,CNT3,CNT2,CNT1,CNT0
OUTPUT       CNTZRN
END

```

Document 1. Definition document for a composite module

DOCUMENT	ADDR8	PAGE	1	LINE	1
*ADDR8	8 bit ADDRESS REGISTER/CNTR				
TYPE	CNTR8				
CONTROL	LDn, INC, BA, BAn, CLK, DC, GND				
BUS	BUS7, BUS6, BUS5, BUS4, BUS3, BUS2, BUS1, BUS0				
OUTPUT	A7, A6, A5, A4, A3, A2, A1, A0				
OUTPUT	OPEN				
	END				

Document 2. Definition document for a 'copy' of the
module definition document 1

DOCUMENT	PAD	PAGE	1	LINE	1
"PAD	256 word by 12 bit RAM with address register				
COUNTER					
SUBMODES	ADDR8, RAM12				
INPUT	IN11, IN10, IN09, IN08, IN07, IN06				
INPUT	IN05, IN04, IN03, IN02, IN01, IN00				
CONTROL	RD, LDn, DC, INC, BA, BAN, CLK				
BUS	BUS7, BUS6, BUS5, BUS4, BUS3, BUS2, BUS1, BUS0				
OUTPUT	OUT11, OUT10, OUT09, OUT08, OUT07, OUT06				
OUTPUT	OUT05, OUT04, OUT03, OUT02, OUT01, OUT00				
END					

Document 3. Definition document for a composite module on the next higher level. The module defined in document 2 is a submodule of this module.

E. The Integrated Logical Simulator

The logical module simulator permits verification of the logical design of a module prior to its construction. It also can be used to compute expected results for comparison with values measured using the module analyzer. In addition, algorithm development for a device can be carried out using the simulator.

As part of an integrated logical design facility, the simulator uses the common logical module structure for description of the module being simulated. The module is defined in terms of its boundary elements and its constituent submodules. These submodules may themselves be composite modules, or they may be uses of previously defined modules. At the lowest level, these are primitive modules with associated programs to perform their function. It is also possible to replace a module in the simulation with the physical device using the module analyzer. In this mode, signals to and from the module analyzer are used with those produced through simulation of the remaining modules.

The simulation is performed at discrete time steps. For each step, the processing consists of computing the outputs of the module from given inputs and controls. All output computation actually takes place at the lowest, or primitive level. Input and control signals are distributed from higher level modules down through submodules until primitive modules are reached. Signals are collected at the boundary of each primitive module until those needed to compute outputs are available; then outputs are computed. These outputs are then distributed throughout the module structure wherever they are needed as inputs or control signals. The processing for a step is complete when all primitive modules have computed their outputs.

A logical module is described to the simulator by specifying its boundary elements and either identifying the module type from a library of previously defined modules or by listing the constituent submodules. The submodule descriptions are in the same form: either uses of previously defined modules or sets of interconnected modules. The boundary specification for each module is ordered by position. The use of the boundary element (Input, Output, Control, Bus) is given explicitly. The signal name given for the element defines the interconnection structure between submodules and to the boundary of the composite module.

Primitive modules are defined for simulation by providing an executable subroutine which computes the outputs of the module from its inputs and controls. In addition, a table must be provided which describes the use made of each boundary element of the module, together with the amount of memory which the module contains. Each such module can have up to three separate subroutines. This permits computation of some outputs based on the presence of only a partial subset of the inputs. This simplifies the processing of modules containing memory elements whose state is to be used during the current step and updated for use in the following step. One subroutine makes their outputs available as soon as the appropriate control signals are present, another updates their internal state when the input data is available.

A two-part, parallel data structure is used for the logic simulation. The first part, the interconnect table (Table 3.1), represents the logical interconnections of the boundary of a module with the boundaries of its constituent submodules. This structure provides the paths for transfer of logic values from their source to all modules where they are used. Each module entry in this structure has a set of pointers to the entries for each of its submodules and a pointer entry for each boundary element of the module and of each of its submodules. These pointer entries are linked together in circular lists connecting all boundary points which share a common signal name. Primitive module entries in this structure contain the subroutines for computing their outputs. If the same module is used more than once, only one copy of it is needed in this structure since its connections to other modules for each use are described at the next higher level.

The second structure used for simulation is a state table which maintains the record of the current state of the module (Table 3.2). This table includes the memory for each primitive module and the current state of all its boundary elements. At higher levels, this table contains only a list of pointers to the entries for each submodule. Since identical modules need not have identical states, this table has separate entries for each use of a module.

These data structures are created from the text description of a module in a two-stage process. Each composite module is first processed by the logic simulation assembler to produce its interconnect table, which shows all logic variables present on its boundary as well as all interconnections between the submodules which make up this module. The output of the logic

assembler for a module is this interconnect table and a list of the sub-modules referenced. When a module is to be simulated, the interconnect tables for all modules contained in it are loaded along with primitive module code and linked together. At the same time, the state table is constructed and the initial variable state filled in. This completes the second stage of the creation of the data structures needed and is followed by the actual simulation steps.

Table 3.1 Interconnect Table Formats for a Module

a. Nonprimitive Module

<u>Word #</u>	<u>Contents</u>
0	# submodules + 1 (M+1)
1-M	Pointer to each submodule entry (*-displacements)
M+1	# boundary elements for module (N)
M+2 - M+N+2	Pointer entry for each boundary element
M+N+3 - END	Entries for submodules
	A (interconnect table for submodule) filled in during second phase.
	One-word pointer entry for each boundary element of submodule, contains submodule index/pin index of next use of signal. Each element entry, whether on the boundary of the module or some submodule, is a circular list link of the form: submodule index/element index where submodule index 0 refers to the boundary of the module. All elements with the same signal name are linked together in a ring.

b. Primitive Module

Word 0:	length of state table entry for module
Word 1:	A (source for state table entry)
Word 2:	A (primary processing code)
Word 3:	control entry 1 A(first control processing code)
Word 4:	control entry 2 A(second control processing code)

A primitive module has up to three processing sections. Each section has its own entry point and is executed according to the following rules:

1. Entry point given at module base + 2. Standard processing. Executed when the specified number of scheduling inputs and controls are valid.
2. Entry at module base + 3: Executed when control with subtype 1 is valid.
3. Entry at module base + 4: Executed when a control with subtype 2 is valid and low or a control with subtype 3 is valid and high.

Table 3.2 State Table Formats

a. State Table Nonprimitive Module Entry

Words 1 to N-1: Address of state table entry for each submodule.

Word N: #1 address of state table entry for last submodule.

b. State Table Primitive Module Entry

Word 0: #2

S	# needed	R	# received
---	----------	---	------------

15 14 8 7 6 0

R and S are set when control entry 1 or 2 respectively can be executed.

received is a count of the number of signals which have their schedule bit set which have been received.

needed is the number of these signals which must be present before the primary entry can be executed.

Words 1 to $\frac{N}{2}$: [#1]

T	ST	S	V	N	C	T	ST	S	V	N	C
---	----	---	---	---	---	---	----	---	---	---	---

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

One byte for each signal:

T = Type 00 = Control
 01 = Output
 10 = Input
 11 = Bus

ST = Subtype Output Bus

00 = Combinatorial/Ungated
 10 = Latched/Ungated
 01 = Combinatorial/enabled
 11 = Latched/enabled

Control

00 = Other
 01 = Immediate entry 1
 10 = Active low - entry 2
 11 = Active high - entry 2

S = Schedule bit - increment received count when this signal is received

V = Valid - set when signal is valid

N = Next Value - retains value for use in clocked outputs

C = Current Value - state of signal if V bit is set.

The last signal word has its flag 1 set. If the module requires memory, it will be located immediately after the signal entries.

F. The Integrated Module Analyzer

The Module Analyzer (MA) performs post construction verification of the logical and engineering design. At this stage in the design the modules involved should be logically correct. However, engineering considerations may affect the performance and implementation of the logical design. Timing characteristics, signal transmission paths, power requirements, and noise environment are among the possible engineering design considerations that may affect the final form of the hardware.

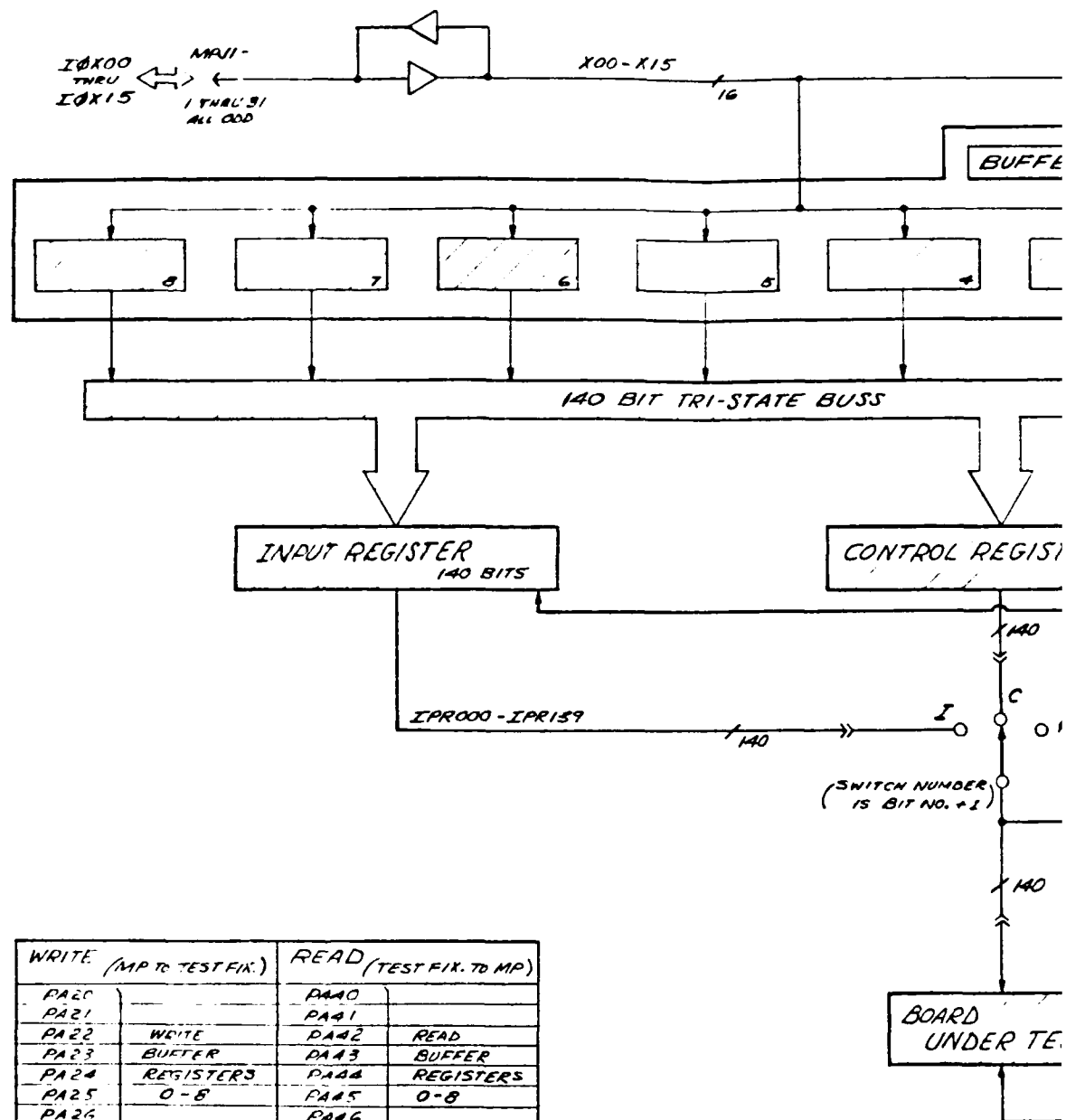
The realization of the prototype MA was influenced by two factors:

(1) the desire to have its elements under direct external software control, and (2) to have a working prototype quickly available. These two factors resulted in a device that has overall structural simplicity. The final implementation of the MA may involve more sophistication that could allow more automation of hardware checkout. For example, the manual switches used for sampling selection could be electronic switches under program control. Nevertheless the prototype MA has proven itself a useful tool. The remainder of this section documents the MA and contains the following paragraphs: Paragraph 1 is a functional description of the entire unit including each of its major registers. Paragraph 2 describes how to operate the MA using the manual controls, external connections, and programmatic capabilities. Para. 3 is a detailed circuit description. Appendices include sample microprograms, and a complete set of drawings.

1. Functional Description

The main features of the MA are depicted in the block diagram of Figure 3.1. Generally, the purpose of the MA is to provide both static and dynamic inputs to a board under test and then to sample the board's outputs after a specified time period. This time period is specified by external control, so that a sequence of tests can determine when an output changes within 5 ns. External addresses and data are supplied on PAX00* to PAX05* and I0X00 to I0X15, respectively. The internal registers are addressed according to the decoding scheme included in Figure 3.1.

The Buffer Register (BR) has several purposes. It serves as a converter from (or to) external 16-bit data to (or from) internal 140-bit data. This is accomplished by nine separately addressable 16-bit registers within BR. After data has been transferred into BR, the information may then be loaded into either the Input Register or the Control Register, both 140 bits.



WRITE (MP TO TEST FIX.)		READ (TEST FIX. TO MP)	
PA20		PA40	
PA21		PA41	
PA22	WRITE	PA42	READ
PA23	BUFFER	PA43	BUFFER
PA24	REGISTERS	PA44	REGISTERS
PA25	0-8	PA45	0-8
PA26		PA46	
PA27		PA47	
PA30		PA50	
PA31	LOAD COUNTER	PA51	READ STATUS
PA32	LOAD INPUT R		
PA33	LOAD CNTRL R		
PA34	RESET		

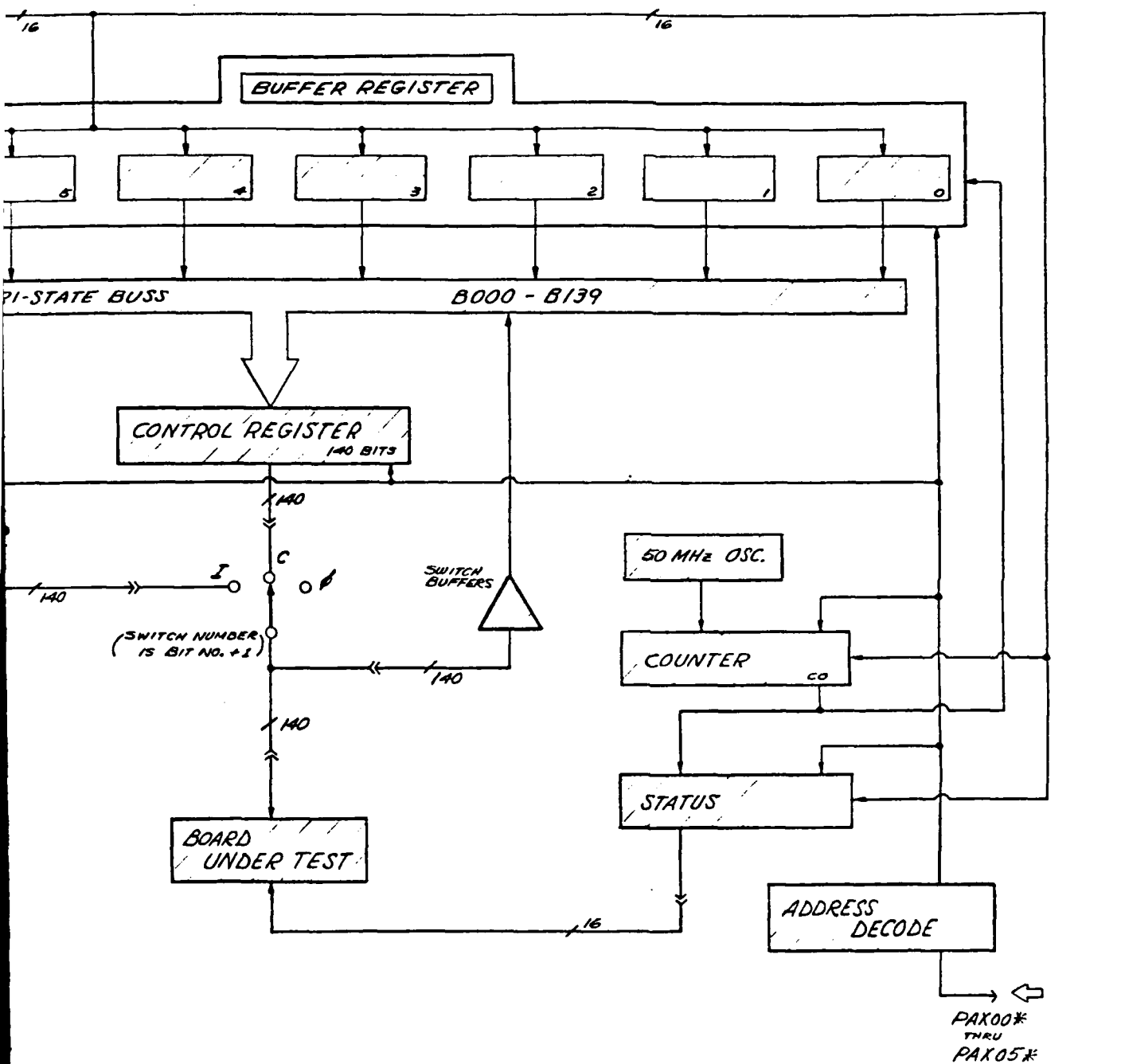


Figure 3.1

TOLERANCES (EXCEPT AS NOTED)	CHI CULLER-HARRISON INC		
DECIMAL	MODULE ANALYZER	SCALE	DRAWN BY S. J. STUBBS
FRACTIONAL	TITLE	APPROVED BY	
ANGULAR	DATE	DRAWING NUMBER	
	7/27/70	MA01	38

BR also serves as the data storage register when the board under test is sampled.

The Input Register (IPR) provides static input data to the board under test. It is a 140-bit register that is loaded from BR. Its outputs may be connected to the board under test by switch settings.

The Control Register (CØN) provides dynamic input data to the board under test. The clocking of CØN provides an initiate pulse to the MA that begins the test sequence. This initiate pulse defines the time of change of any bit in CØN and also provides a start pulse to the counter.

The Counter consists of two elements, the Count Save (CS) register and a 50 Mhz up-counting register (CNT). The counter has two functions: It generates timing strobes for the various data transfer operations to, from, and within the MA. It also provides the sample strobe that defines when data from the board under test is loaded into BR and the Status Register. The length of time between the initiate pulse of CØN and the sample pulse of CNT is determined by the number loaded into CS. This time may be practically varied from 30 ns to 327 µs in 5 ns intervals.

The Status Register (STAT) is a 16-bit register whose inputs are directly attached to external probes. These probes may be connected to any signal on the board under test; this provides the capability to sample 16 internal states in addition to the I/O pins. Data from STAT is directly read from the X-Bus since it is only 16 bits wide.

Switches on the MA are set to correspond to the function of each pin. Input pins are switched to "I" or "C" depending on whether the input is a static or dynamic one. At sample time the state of these inputs is loaded into BR. Output pins are switched to "Ø" so that at sample time the state of the output pin is recorded. Power and ground pins are switched to "Ø"; power wiring is not a logic input so it is wired separately to the board connector.

2. Operation

The operation of the MA is governed by three conditions: (1) External connections, (2) switch settings, and (3) control of external logic signals.

The external connections must be made prior to powering on. The power cable must be plugged to both the MA and an external 5V power supply. A 50-pin flat cable must be connected to Connector MP1. This provides external input/output; pin assignments are given in Table 3.3. The opposite end is connected to the device providing control. The board to be tested should be

Table 3.3 MP1 Pin Assignments

<u>Pin</u>	<u>Signal</u>	<u>Pin</u>	<u>Signal</u>
1	I0X00*	26	I0X12*R
2	I0X00*R	27	I0X13*
3	I0X01*	28	I0X13*R
4	I0X01*R	29	I0X14*
5	I0X02*	30	I0X14*R
6	I0X02*R	31	I0X15*
7	I0X03*	32	I0X15*R
8	I0X03*R	33	I0XENA*
9	I0X04*	34	I0XENA*R
10	I0X04*R	35	UNUSED
11	I0X05*	36	UNUSED
12	I0X05*R	37	PAX00*
13	I0X06*	38	PAX00*R
14	I0X06*R	39	PAX01*
15	I0X07*	40	PAX01*R
16	I0X07*R	41	PAX02*R
17	I0X08*	42	PAX02*R
18	I0X08*R	43	PAX03*
19	I0X09*	44	PAX03*R
20	I0X09*R	45	PAX04*
21	I0X10*	46	PAX04*R
22	I0X10*R	47	PAX05*
23	I0X11*	48	PAX05*R
24	I0X11*R	49	UNUSED
25	I0X12*	50	UNUSED

43

placed in the edge connector provided on the top of the MA. When these connections have been made the power-on switch may be enabled.

Before programmatic manipulation is begun, the desired switch settings should be made. Output pins of the board under test should have their corresponding switches set to "0". Inputs should be set to "I" or "C" depending on whether the pin will have a static or dynamic input. In addition to switch settings the user may desire to attach any or all of the 16 probes to internal points on the board.

Programmatic control of the MA is achieved by manipulation of the address (PAX00* to PAX15*), data (I0X00 to I0X15), and enable (I0XENA*) signals. Generally the procedure to perform a test sequence is as follows:

- (1) Reset the MA (PA = 34₉).
- (2) Load CS with the desired 16-bit count (PA = 31₉).
- (3) Load BR with the desired data for IPR (PA 20 to 30).
- (4) Transfer BR to IPR (PA32).
- (5) Load BR with the desired data for C0N (PA 20 to 30).
- (6) Transfer BR to C0N (PA33); this initiates the count previously loaded into CS.
- (7) Wait for sample pulse.
- (8) Read BR (PA 40 to 50).
- (9) Read STAT (PA 51).

These data transfers are controlled by internally generated timing strobes. These strobes define the times during which commands and data are to be supplied to the MA, and they define when MA output data is to be externally sampled. The timing of input to the MA is given in Figure 3.2. Both I0X (data) and address (PA*) should be changed a minimum of 100 ns before making I0XENA* active. I0XENA* enables the X-Bus, and 820 ns later the timing pulse, STR0BE1*, is activated. In Figure 3.2 the signal REG DECODE represents the register enable signal that is a decode of PA 20-34. The combination of decode and STR0BE1* generates the appropriate register clock. This latches the data in the corresponding register. The timing is the same for clocking external data into a 16-bit segment of BR, transferring 140 bits from BR to IPR or C0N, and generating a reset pulse. The user must simply keep I0XENA* active for more

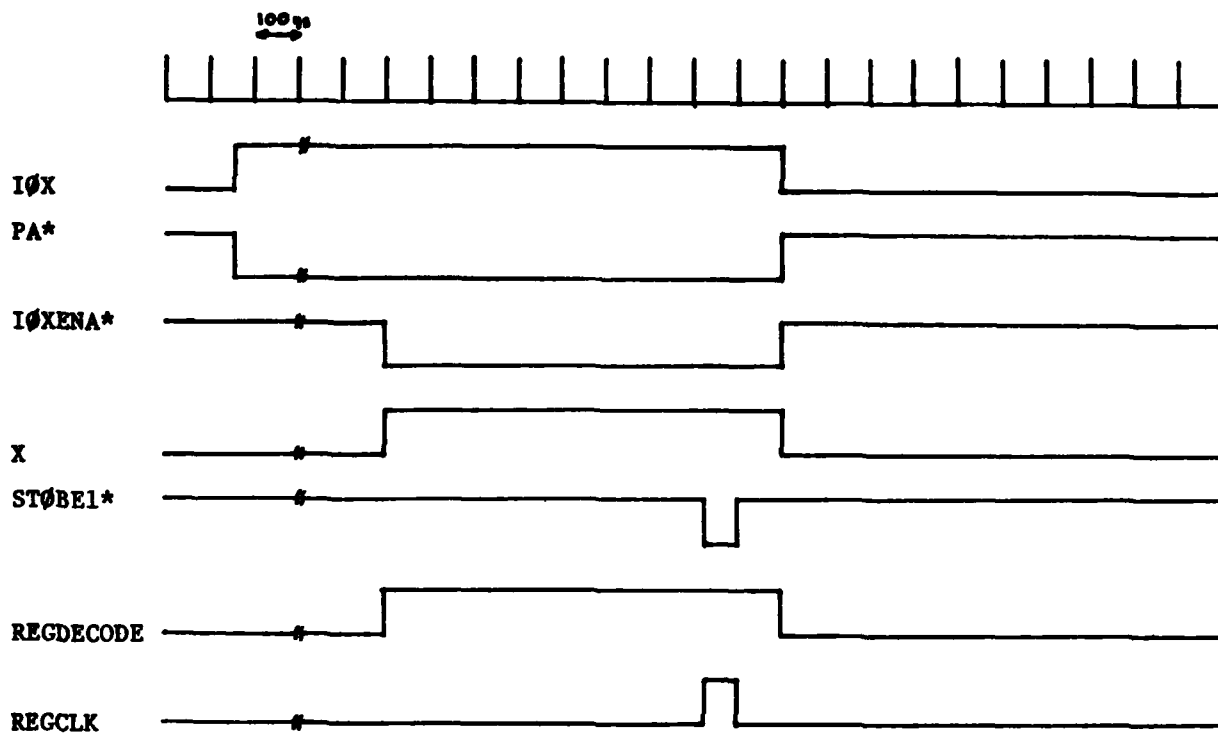


Figure 3.2 Input Timing Diagram

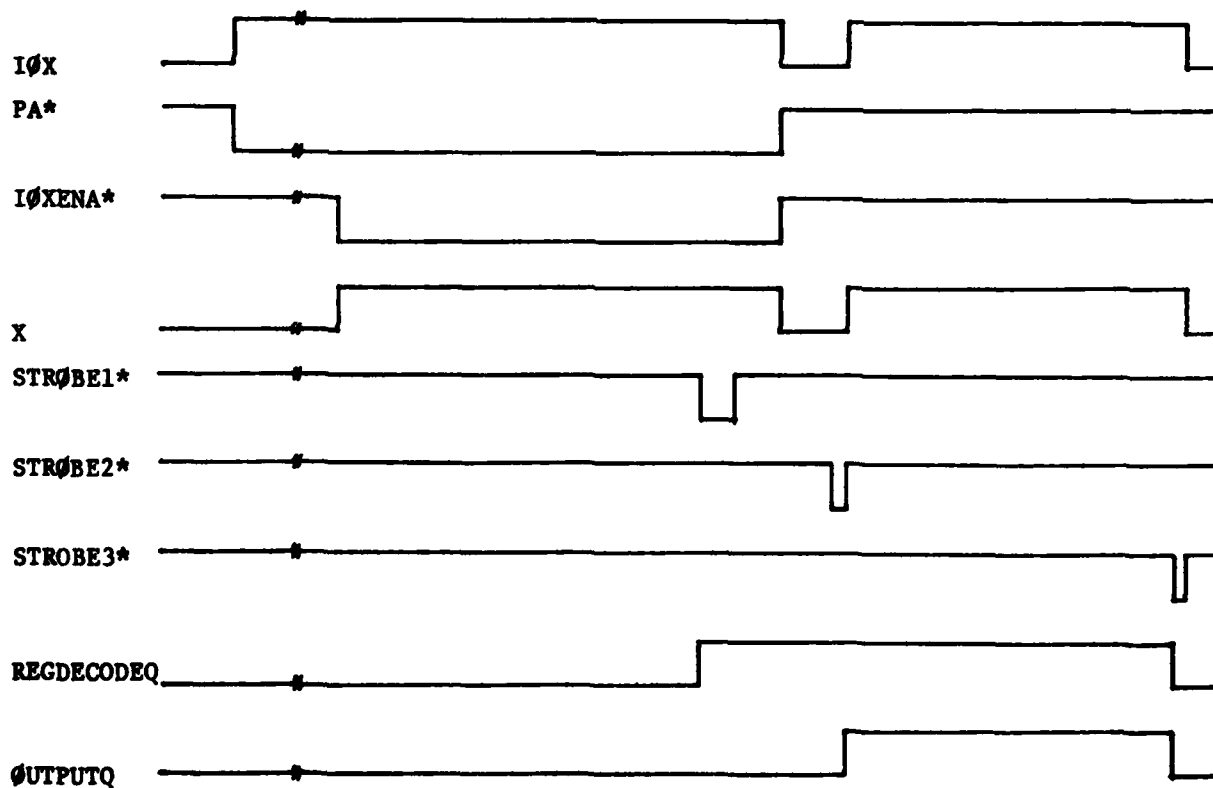


Figure 3.3 Output Timing Diagram

than 900 ns and then data, address and IØXENA* may be made inactive simultaneously.

The timing for output from the MA is similar to the input timing with the addition that the output data must be sampled during a specified time period. Output timing is given in Figure 3.3. The timing for output is the same as input thru the generation of STRØBE1*. This allows the MA to decode the address of the register being selected for output. However, after STRØBE1* there are additional timing rules. The user must stop driving the enable, address, and data signals before STRØBE2* which occurs at 1140 ns. At this time the MA will begin driving the IØX lines and will continue until STRØBE3* at 1920 ns. Data must be sampled between 1140 and 1920 ns.

Appendix 1 gives sample micro programs for controlling the MA. They are written for a CHI MP32A Macro Processor. The instruction ØUTPUT CLR provides proper timing to IØXENA*, and INPUT samples IØX during the time it is driven by the MA. These micro routines assume that PD (which drives IØX) has been previously set up with the desired data.

3. Circuit Description

This section will first describe each of the major registers within the MA and then describe the control circuits that generate the proper timing and event sequence. Refer to the logic drawings in Appendix 2.

The Buffer Register (BR) is the only register that functions as both an array of 16-bit registers and as a 140-bit register. BR is made of DM8542 4-bit three-state registers. External input data (IØX00 to IØX15) is received by 8T95 three-state drivers whose outputs are connected to the A-Bus of the 8542. The 140 bits are segmented into 16-bit groups, each group having a separate input and separate output enable controls. These controls are direct decodes of the PA* signals. Although the entire 140-bit register must have a common clock to allow loading of 140 data bits from the B-Bus, sixteen bit loading is accomplished by the A-Bus controls. Thus BR is externally read in 16-bit segments.

When data is to be transferred from BR to either the Input or Control Registers the B-Bus output enable control is activated. Data is clocked into these registers at STRØBE1* time. Both the Input and Control Registers (IPR and CØN) are made of SN74LS174 6-bit registers.

The outputs of IPR and ~~CON~~ may be switched to the board under test and in this case they are buffered from the B-Bus of the 8542 by three-state buffers (SN74367). When data is to be loaded into BR from this side the input control LDBR* activates both the three-state buffers and the B-Bus input of BR. In this case BRCLK is generated from the counter.

The Status Register is made of AM25S18 4-bit three-state registers. Its inputs are directly attached to the external probes. It is clocked at the sample time generated by the counter. Its three-state outputs are enabled onto the X-Bus when its address has been decoded. It is read in the same manner as any of the 16-bit segments of BR.

Timing pulses and control signals are generated by a 16-bit up counter. The basic clock of the counter is supplied by a 50 Mhz oscillator. The outputs of the counter are used in two functionally different ways and divide into two phases. The first phase generates all the timing and control signals; the second phase generates the data sample pulse which may be varied between 30 ns and 327 μ s in 5 ns intervals.

The first phase generally begins with the activation of I~~X~~ENA* and lasts until STR~~O~~BE3*. The only exception to this is when the data sampling pulse is to be generated (see below).

When I~~X~~ENA* becomes active it clocks CNTENAQ which enables the 16-bit counter (AM93S16). CNTENAQ is also wired to the reset input of each stage of the counter to ensure the count begins at zero. The counter is a synchronous counter, but the carryout of a stage is clocked into the next stage with a delay of one clock. This was necessary because the delay time from clock to carryout plus the setup time from carry into clock is longer than the 20 ns clock period generated by the oscillator. Thus it was necessary to stretch the carryout by two inverters and an ~~OR~~ gate in order to meet the setup requirements.

The strobe pulses, STR~~O~~BE1-3* are generated by simply using an 8-input AND gate with the appropriate inputs. STROBE1* generates all the 140-bit register clocks: C~~O~~UNTCLK, IPRCLK, C~~O~~NCLK, and BRCLK. STR~~O~~BE2* clocks ~~O~~UTENAQ, the enable pulse for MA output. STR~~O~~BE3* is generally used to reset control flip flops to their original state so that each time the MA is addressed it begins operation from the same state. It resets ~~O~~UTPUTQ, ~~O~~UTENAQ, LDBR and CNTENAQ.

The exception to this sequence is when the counter is used to generate the sample pulse, T0. The decode of PA33 (Load Control Register) generates the signal C0NCLK at STR0BE1* time. C0NCLK alters the usual sequence of events. Referring to Figure 3.4 the first events are the activation of STRINHQ* and LDBR*. STRINHQ* disables STROBE2-3*; LDBR* enables the B-Bus input of BR. C0NCLK* is used to enable the parallel load feature of the 93S16 counter. So the contents of CS are loaded into the counter at every DELCLK until C0NCLK* goes inactive. At this trailing edge of C0NCLK the counter begins to count and the Control Register is clocked. Hence the dynamic inputs to the board under test change at the same time the counter begins to count.

When the counter reaches a count of all ones the sample pulse, T0, is generated. T0 generates a BRCLK, so the data from the board under test (or the contents of other registers as determined by the switch settings) is loaded into BR. T0 also clocks data into the Status Register. Additionally T0 resets STRINHQ*, so that the counter now begins counting again from zero. Therefore, the generation of the STR0BE1-3* pulses proceeds as previously described with STR0BE3* being the only important event. STR0BE3* resets any remaining active pulses so that the MA is returned to its idle state.

The implementation of selectable 5 ns delays is accomplished by using a delay line with 2-1/2 ns delay taps. Delays of 5, 10, 15, and 20 ns are used to provide four selectable clocks. DELO is selected at all times except when LDBR is active. When LDBR is active the delay is selected by the two least significant bits of CS.

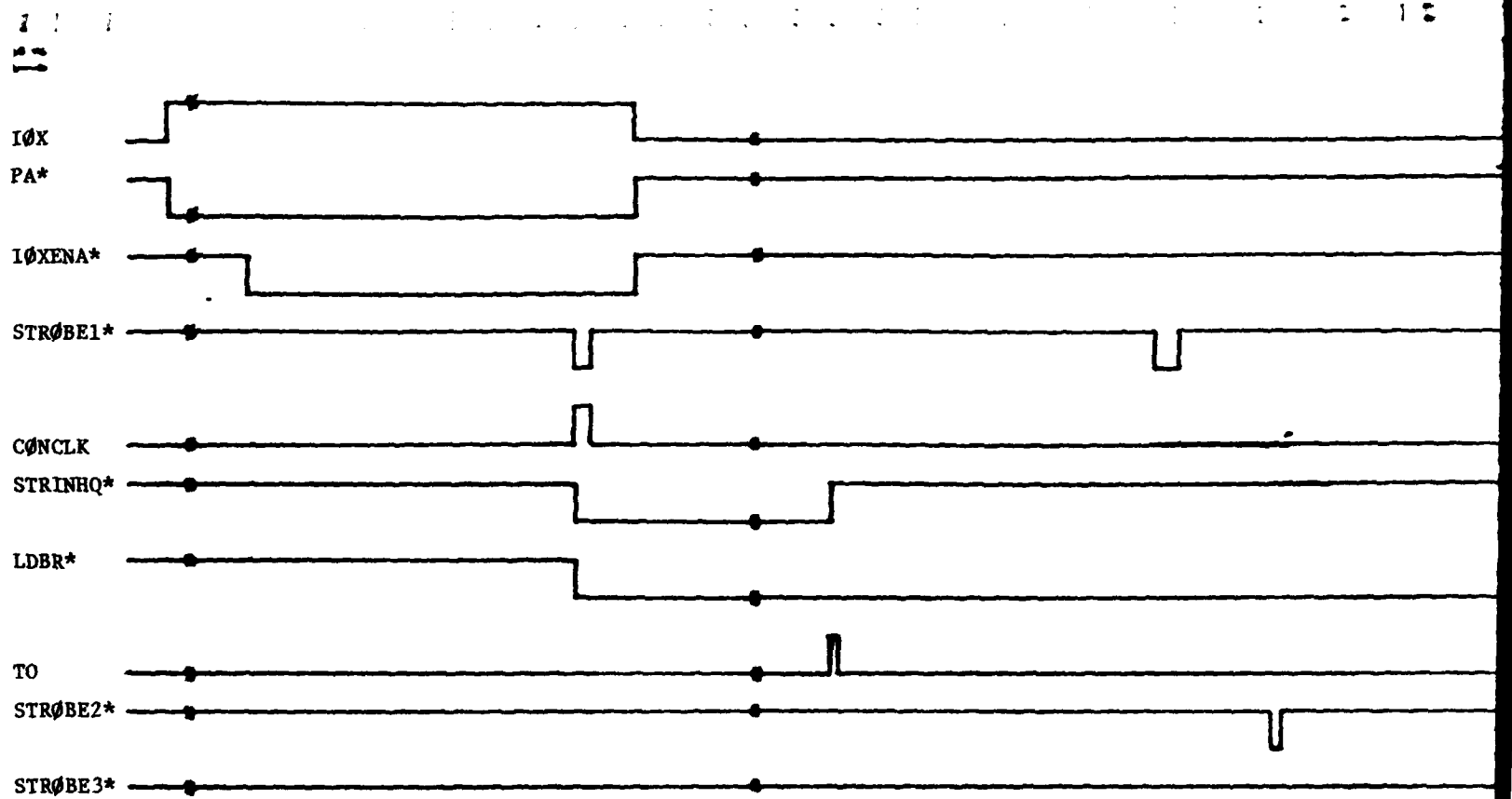


Figure 3.4 Test Sequence Timing

APPENDIX A: SAMPLE OUTPUTS FROM THE STRUCTURED
DESIGN SYSTEM

DISCOUNT connAPU
*connAPU

PAGE

1

LINE

1

P	*Part name
1	TSel
2	TCLK
3	XSel
4	XMCir
5	@100
6	@101
7	@102
8	@103
9	@104
10	@105
11	@106
12	@107
13	@108
14	@109
15	@110
16	@111
17	GASel
18	GACir
19	OC0
20	OC1
21	OC2
22	OBSel
23	OBCir
24	UCLK
25	USel
26	MPCLK
27	MXCLK
28	MYCLK
29	MPRND
30	MXSel
31	MXCir
32	MYSel
33	MYCir
34	CLK
35	YMSel
36	YMCir
37	@200
38	@201
39	@202
40	@203
41	@204
42	@205
43	@206
44	@207
45	@208
46	@209
47	@210
48	@211
49	HASel
50	HACir
51	HC0
52	HC1
53	HC2
54	HBSel
55	HBCir
56	VCLK
57	VSel
58	NC

DOCUMENT connapu

PAGE

1 LINE

61

59 G11
60 GCIN
61 HCRY
62 GND
63 GND
64
65 I100
66 I101
67 I102
68 I103
69 I104
70 I105
71 I106
72 I107
73 I200
74 I201
75 I202
76 I203
77 I204
78 I205
79 I206
80 I207
81 I208
82 I209
83 I210
84 I211
85 I300
86 I301
87 I302
88 I303
89 I304
90 I305
91 I306
92 I307
93 I308
94 I309
95 I310
96 I311
97 M0
98 M1
99 M2
100 M3
101 M4
102 HB0
103 HB1
104 HAO
105 HA1
106 HBP0
107 HBP1
108 HBP2
109 OB0
110 OB1
111 GAO
112 GA1
113 GBP0
114 GBP1
115 GBP2
116 UO
117 VO
118

DOCUMENT	connAPU	PAGE	1	LINE	121
119	NC				
120					
121	VCC				
122	VCC				
123					
124					
125					
126					
127					
128					
\$					

DOCUMENT	manuAPU	PAGE	1	LINE	1
Submod.	Physical	Display			
Name	Location	Location			
H	J19	0,0			
MX1	J10	0,0	MX2	L10	0,40
MX3	N10	30,25			
MY1	J1	0,0	MY2	L1	0,40
MY3	N1	30,25			
T1	C10	0,0	T2	G10	0,40
F1	A10	30,0	F2	E10	30,40
G1	C19	0,0	G2	C30	0,40
G3	C41	30,25			
GA1	A21	0,0	GA2	A32	0,40
GA3	A43	30,25			
GB1	E21	0,0	GB2	E32	0,40
GB3	F43	30,25			
H1	R19	0,0	H2	R30	0,40
H3	R41	30,25			
HA1	N21	0,0	HA2	N32	0,40
HA3	N43	30,25			
HB1	T21	0,0	HB2	T32	0,40
HB3	T43	30,25			
U1	G21	0,0	U2	G32	0,40
U3	O43	30,25			
V1	V21	0,0	V2	V32	0,40
V3	V43	30,25			
XM1	C1	0,0	XM2	E1	0,40
XM3	G1	30,25			
YM1	R1	0,0	YM2	T1	0,40
YM3	V1	30,25			
DCD4	A1	0,0	DCD5	R10	0,40
DCD6	T10	30,0	DCD7	V10	30,40
resistAPU	G42	0,0			

S

	64	3-Part	name
I	K4	1	MX07 J19
I	K5	2	MX06 J20
I	K6	3	MX05 J21
I	K7	4	MX04 J22
I	K8	5	MX03 J23
I	K9	6	MX02 J24
I	K10	7	MX01 J25
I	K11	8	MX00 J26
0	PR22	9	MPL00 J27
0	PR21	10	MPL01 J28
0	PR20	11	MPL02 J29
0	PR19	12	MPL03 J30
0	PR18	13	MPL04 J31
0	PR17	14	MPL05 J32
0	PR16	15	MPL06 J33
0	PR15	16	MPL07 J34
0	PR14	17	MPL08 J35
0	PR13	18	MPL09 J36
0	PR12	19	MPL10 J37
0	SGNL	20	OPEN J38
C	TRIL	21	TRUE J39
C	TRIM	22	TRUE J40
G	GND	23	GND J41
G	GND	24	GND J42
G	GND	25	GND J43
G	GND	26	GND J44
C	CLKL	27	MPCLK J45
C	CLKM	28	MPCLK J46
0	PR11	29	MPF00 J47
0	PR10	30	MPF01 J48
0	PR9	31	MPF02 J49
0	PR8	32	MPF03 J50
0	PR7	33	MPF04 L50
0	PR6	34	MPF05 L49
0	PR5	35	MPF06 L48
0	PR4	36	MPF07 L47
0	PR3	37	MPF08 L46
0	PR2	38	MPF09 L45
0	PR1	39	MPF10 L44
0	SGNM	40	MPF11 L43
I	NC	41	OPEN L42
I	Y5GN	42	MY11 L41
I	Y1	43	MY10 L40
I	Y2	44	MY09 L39
I	Y3	45	MY08 L38
I	Y4	46	MY07 L37
I	Y5	47	MY06 L36
V	VCC	48	VCC L35
V	VCC	49	VCC L34
V	VCC	50	VCC L33
I	Y6	51	MY05 L32
I	Y7	52	MY04 L31
I	Y8	53	MY03 L30

*K11
25LS157 16 2*Part name

C	SEL	1	MXSel	J10
I	1A	2	I208	J11
I	1B	3	I308	J12
O	1Y	4	MX08	J13
I	2A	5	I209	J14
I	2B	6	I309	J15
O	2Y	7	MX09	J16
G	GND	8	GND	J17
O	3Y	9	MX10	K17
I	3B	10	I310	K16
I	3A	11	I210	K15
O	4Y	12	MX11	K14
I	4B	13	I311	K13
I	4A	14	I211	K12
C	STROBE	15	MXC1r	K11
V	VCC	16	VCC	K10

*MX2
25LS157 16 2*Part name

C	SEL	1	MXSel	L10
I	1A	2	I204	L11
I	1B	3	I304	L12
O	1Y	4	MX04	L13
I	2A	5	I205	L14
I	2B	6	I305	L15
O	2Y	7	MX05	L16
G	GND	8	GND	L17
O	3Y	9	MX06	M17
I	3B	10	I306	M16
I	3A	11	I206	M15
O	4Y	12	MX07	M14
I	4B	13	I307	M13
I	4A	14	I207	M12
C	STROBE	15	MXC1r	M11
V	VCC	16	VCC	M10

*MX3
25LS157 16 2*Part name

C	SEL	1	MXSel	N10
I	1A	2	I200	N11
I	1B	3	I300	N12
O	1Y	4	MX00	N13
I	2A	5	I201	N14
I	2B	6	I301	N15
O	2Y	7	MX01	N16
G	GND	8	GND	N17
O	3Y	9	MX02	P17
I	3B	10	I302	P16
I	3A	11	I202	P15
O	4Y	12	MX03	P14
I	4B	13	I303	P13
I	4A	14	I203	P12
C	STROBE	15	MXC1r	P11
V	VCC	16	VCC	P10

IO = Selectable Input

----- C = ^{CONTROL}
~~CLOCK~~

I = INPUT

O = OUTPUT

G = GROUND

V = Voltage

```

*MY1
25LS157 16 2*Part name
C Sel 1 MYSel J1
I 1A 2 I200 J2
I 1B 3 I300 J3
O 1Y 4 MY00 J4
I 2A 5 I209 J5
I 2B 6 I309 J6
O 2Y 7 MY09 J7
G GND 8 GND J8
O 3Y 9 MY10 K8
I 3B 10 I310 K7
I 3A 11 I210 K6
O 4Y 12 MY11 K5
I 4B 13 I311 K4
I 4A 14 I211 K3
C STROBE 15 MYC1r K2
V VCC 16 VCC K1

```

```

*MY2
25LS157 16 2*Part name
C Sel 1 MYSel L1
I 1A 2 I204 L2
I 1B 3 I304 L3
O 1Y 4 MY04 L4
I 2A 5 I205 L5
I 2B 6 I305 L6
O 2Y 7 MY05 L7
G GND 8 GND L8
O 3Y 9 MY06 M8
I 3B 10 I306 M7
I 3A 11 I206 M6
O 4Y 12 MY07 M5
I 4B 13 I307 M4
I 4A 14 I207 M3
C STROBE 15 MYC1r M2
V VCC 16 VCC M1

```

```

*MY3
25LS157 16 2*Part name
C Sel 1 MYSel N1
I 1A 2 I200 N2
I 1B 3 I300 N3
O 1Y 4 MY00 N4
I 2A 5 I201 N5
I 2B 6 I301 N6
O 2Y 7 MY01 N7
C GND 8 GND N8
O 3Y 9 MY02 P8
I 3B 10 I302 P7
I 3A 11 I202 P6
O 4Y 12 MY03 P5
I 4B 13 I303 P4
I 4A 14 I203 P3
C STROBE 15 MYC1r P2
V VCC 16 VCC P1

```



```

*T1
2SL509 16 2*Part name
C S 1 TSe1 C10
G G0 2 T04 C11
I DOA 3 F04 C12
I DOB 4 I104 C13
I D1B 5 I105 C14
I D1A 6 F05 C15
G Q1 7 T05 C16
G GND 8 GND C17
C CP 9 TCLK D17
G Q2 10 T06 D16
I D2A 11 F06 D15
I D2B 12 I106 D14
I D3B 13 I107 D13
I D3A 14 F07 D12
G Q3 15 T07 D11
V VCC 16 VCC D10

```

```

*T2
2SL509 16 2*Part name
C S 1 TSe1 G10
G G0 2 T00 G11
I DOA 3 F00 G12
I DOB 4 I100 G13
I D1B 5 I101 G14
I D1A 6 F01 G15
G Q1 7 T01 G16
G GND 8 GND G17
C CP 9 TCLK H17
G Q2 10 T02 H16
I D2A 11 F02 H15
I D2B 12 I102 H14
I D3B 13 I103 H13
I D3A 14 F03 H12
G Q3 15 T03 H11
V VCC 16 VCC H10

```

```

*F1
2SL52517 20 2
I A1 1 GA11 A10
I B1 2 T05 A11
I A0 3 GA11 A12
I B0 4 T04 A13
C S0 5 G00 A14
C S1 6 GC1 A15
C S2 7 GC2 A16
G F0 8 F04 A17
G F1 9 F05 A18
G GND 10 GND A19
G F2 11 F06 B19
G F3 12 F07 B18
G GVR 13 OPEN B17
G C0 14 OPEN B16
I Cn 15 F2C0 B15
I B3 16 T07 B14
I A3 17 GA11 B13
I B2 18 T06 B12
I A2 19 GA11 B11
V VCC 20 VCC B10

```

```

*F2
2SL52517 20 2*Part name
I A1 1 GA11 E10
I B1 2 T01 E11
I A0 3 GA11 E12
I B0 4 T00 E13
C S0 5 G00 E14
C S1 6 GC1 E15
C S2 7 GC2 E16
G F0 8 F00 E17
G F1 9 F01 E18
G GND 10 GND E19
G F2 11 F02 F19
G F3 12 F03 F18
G GVR 13 OPEN F17
G C0 14 F2C0 F16
I Cn 15 G1C0 F15
I B3 16 T03 F14
I A3 17 GA11 F13
I B2 18 T02 F12
I A2 19 GA11 F11
V VCC 20 VCC F10

```

*G1
25LS2517 20 2*Part name
I A1 1 GA09 C19
I B1 2 GB09 C20
I A0 3 GA08 C21
I B0 4 GB08 C22
C S0 5 GC0 C23
C S1 6 GC1 C24
C S2 7 GC2 C25
C F0 8 GC8 C26
C F1 9 GC9 C27
C GND 10 GND C28
C F2 11 G10 D28
C F3 12 G11 D27
C QVR 13 QPEN D26
C C0 14 G1C0 D25
I Cn 15 G2C0 D24
I B3 16 GB11 D23
I A3 17 GA11 D22
I B2 18 GB10 D21
I A2 19 GA10 D20
V VCC 20 VCC D19

*G2
25LS2517 20 2*Part name
I A1 1 GA05 C30
I B1 2 GB05 C31
I A0 3 GA04 C32
I B0 4 GB04 C33
C S0 5 GC0 C34
C S1 6 GC1 C35
C S2 7 GC2 C36
C F0 8 GC4 C37
C F1 9 GC5 C38
C GND 10 GND C39
C F2 11 GC6 D39
C F3 12 GC7 D38
C QVR 13 QPEN D37
C C0 14 G2C0 D36
I Cn 15 G3C0 D35
I B3 16 GB07 D34
I A3 17 GA07 D33
I B2 18 GB06 D32
I A2 19 GA06 D31
V VCC 20 VCC D30

*G3
25LS2517 20 2*Part name
I A1 1 GA01 C41
I B1 2 GB01 C42
I A0 3 GA00 C43
I B0 4 GB00 C44
C S0 5 GC0 C45
C S1 6 GC1 C46
C S2 7 GC2 C47
C F0 8 GC0 C48
C F1 9 GC1 C49
C GND 10 GND C50
C F2 11 GC2 D50
C F3 12 GC3 D49
C QVR 13 QPEN D48
C C0 14 G3C0 D47
I Cn 15 GCIN D46
I B3 16 GB03 D45
I A3 17 GA03 D44
I B2 18 GB02 D43
I A2 19 GA02 D42
V VCC 20 VCC D41


```

*GA1
25LS157 16 2*Part name
C Sel 1 GAsel A21
I 1A 2 MPF09 A22
I 1B 3 I208 A23
0 1Y 4 GA08 A24
I 2A 5 MPF10 A25
I 2B 6 I209 A26
0 2Y 7 GA09 A27
0 GND 8 GND A28
I 3Y 9 GA10 B28
I 3B 10 I210 B27
I 3A 11 MPF11 B26
0 4Y 12 GA11 B25
I 4B 13 I211 B24
I 4A 14 MPF11 B23
C STR0BE 15 GACIr B22
V VCC 16 VCC B21

```

```

*GA2
25LS157 16 2*Part name
C Sel 1 GAsel A32
I 1A 2 MPF05 A33
I 1B 3 I204 A34
0 1Y 4 GA04 A35
I 2A 5 MPF06 A36
I 2B 6 I205 A37
0 2Y 7 GA05 A38
G GND 8 GND A39
0 3Y 9 GA06 B39
I 3B 10 I206 B38
I 3A 11 MPF07 B37
0 4Y 12 GA07 B36
I 4B 13 I207 B35
I 4A 14 MPF08 B34
C STR0BE 15 GACIr B33
V VCC 16 VCC B32

```

```

*GA3
25LS157 16 2*Part name
C Sel 1 GAsel A43
I 1A 2 MPF01 A44
I 1B 3 I200 A45
0 1Y 4 GA00 A46
I 2A 5 MPF02 A47
I 2B 6 I201 A48
0 2Y 7 GA01 A49
G GND 8 GND A50
0 3Y 9 GA02 B50
I 3B 10 I202 B49
I 3A 11 MPF03 B48
0 4Y 12 GA03 B47
I 4B 13 I203 B46
I 4A 14 MPF04 B45
C STR0BE 15 GACIr B44
V VCC 16 VCC B43

```


*GB1

25LS157

16	2*Part	name
C	Sel	1 GBSel E21
I	1A	2 MPF08 E22
I	1B	3 U08 E23
I	1Y	4 GB08 E24
I	2A	5 MPF09 F25
I	2B	6 U09 E26
I	2Y	7 GB09 F27
G	GND	8 GND E28
I	3Y	9 GB10 F28
I	3B	10 U10 F27
I	3A	11 MPF10 F26
I	4Y	12 GB11 F25
I	4B	13 U11 F24
I	4A	14 MPF11 F23
C	STROBE	15 GBC1r F22
V	VCC	16 VCC F21

*GB2

25LS157

16	2*Part	name
C	Sel	1 GBSel E32
I	1A	2 MPF04 E33
I	1B	3 U04 E34
I	1Y	4 GB04 F35
I	2A	5 MPF05 E36
I	2B	6 U05 E37
I	2Y	7 GB05 E38
G	GND	8 GND E39
I	3Y	9 GB06 F39
I	3B	10 U06 F38
I	3A	11 MPF06 F37
I	4Y	12 GB07 F36
I	4B	13 U07 F35
I	4A	14 MPF07 F34
C	STROBE	15 GBC1r F33
V	VCC	16 VCC F32

*GB3

25LS157

16	2*Part	name
C	Sel	1 GBSel E43
I	1A	2 MPF00 E44
I	1B	3 U00 E45
I	1Y	4 GB00 E46
I	2A	5 MPF01 E47
I	2B	6 U01 E48
I	2Y	7 GB01 E49
G	GND	8 GND E50
I	3Y	9 GB02 F50
I	3B	10 U02 F49
I	3A	11 MPF02 F48
I	4Y	12 GB03 F47
I	4B	13 U03 F46
I	4A	14 MPF03 F45
C	STROBE	15 GBC1r F44
V	VCC	16 VCC F43

```

*H1
25LS2517 20 2*Part name
I A1 1 HA09 R19
I B1 2 HB09 R20
I A0 3 HA08 R21
I B0 4 HB08 R22
C S0 5 HC0 R23
C S1 6 HC1 R24
C S2 7 HC2 R25
C F0 8 HC8 R26
C F1 9 HO9 R27
C GND 10 GND R28
C F2 11 H10 S28
C F3 12 H11 S27
C OVR 13 OPEN S26
C C0 14 HCRY S25
I Cn 15 H2C0 S24
I B3 16 HB11 S23
I A3 17 HA11 S22
I B2 18 HB10 S21
I A2 19 HA10 S20
V VCC 20 VCC S19

```

```

*H2
25LS2517 20 2*Part name
I A1 1 HA05 R30
I B1 2 HB05 R31
I A0 3 HA04 R32
I B0 4 HB04 R33
C S0 5 HC0 R34
C S1 6 HC1 R35
C S2 7 HC2 R36
C F0 8 HO4 R37
C F1 9 HO5 R38
C GND 10 GND R39
C F2 11 HO6 S39
C F3 12 HO7 S38
C OVR 13 OPEN S37
C C0 14 H2C0 S36
I Cn 15 H3C0 S35
I B3 16 HB07 S34
I A3 17 HA07 S33
I B2 18 HB06 S32
I A2 19 HA06 S31
V VCC 20 VCC S30

```

```

*H3
25LS2517 20 2*Part name
I A1 1 HA01 R41
I B1 2 HB01 R42
I A0 3 HA00 R43
I B0 4 HB00 R44
C S0 5 HC0 R45
C S1 6 HC1 R46
C S2 7 HC2 R47
C F0 8 H00 R48
C F1 9 H01 R49
C GND 10 GND R50
C F2 11 H02 S50
C F3 12 H03 S49
C OVR 13 OPEN S48
C C0 14 H3C0 S47
I Cn 15 GND S46
I B3 16 HB03 S45
I A3 17 HA03 S44
I B2 18 HB02 S43
I A2 19 HA02 S42
V VCC 20 VCC S41

```


*HA1

25LS157	16	2*Part	name
C	1	HASe1	N21
I	2	MPL08	N22
I	3	I308	N23
I	4	HA08	N24
I	5	MPL09	N25
I	6	I309	N26
I	7	HA09	N27
G	8	GND	N28
I	9	HA10	P28
I	10	I310	P27
I	10	MPL10	P26
I	12	HA11	P25
I	13	I311	P24
I	14	MPL00	P23
C	15	HAC1r	P22
V	16	VCC	P21

*HA2

25LS157	16	2*Part	name
C	1	HASe1	N32
I	2	MPL04	N33
I	3	I304	N34
I	4	HA04	N35
I	5	MPL05	N36
I	6	I305	N37
I	7	HA05	N38
G	8	GND	N39
I	9	HA06	P39
I	10	I306	P38
I	10	MPL06	P37
I	12	HA07	P36
I	13	I307	P35
I	14	MPL07	P34
C	15	HAC1r	P33
V	16	VCC	P32

*HA3

25LS157	16	2*Part	name
C	1	HASe1	N43
I	2	MPL00	N44
I	3	I300	N45
I	4	HA00	N46
I	5	MPL01	N47
I	6	I301	N48
I	7	HA01	N49
G	8	GND	N50
I	9	HA02	P50
I	10	I302	P49
I	10	MPL02	P48
I	12	HA03	P47
I	13	I303	P46
I	14	MPL03	P45
C	15	HAC1r	P44
V	16	VCC	P43

```

*HB1
25LS157 16 2*Part name
C Sel 1 HBSel T21
I 1A 2 MPF08 T22
I 1B 3 V08 T23
I 1Y 4 HB08 T24
I 2A 5 MPF09 T25
I 2B 6 V09 T26
I 2Y 7 HB09 T27
GND 8 GND T28
I 3Y 9 HB10 U28
I 3B 10 V10 U27
I 3A 10 MPF10 U26
I 4Y 12 HB11 U25
I 4B 13 V11 U24
I 4A 14 MPF11 U23
C STR0BE 15 HBC1r U22
V VCC 16 VCC U21

```

```

*HB2
25LS157 16 2*Part name
C Sel 1 HBSel T32
I 1A 2 MPF04 T33
I 1B 3 V04 T34
I 1Y 4 HB04 T35
I 2A 5 MPF05 T36
I 2B 6 V05 T37
I 2Y 7 HB05 T38
G GND 8 GND T39
I 3Y 9 HB06 U39
I 3B 10 V06 U38
I 3A 10 MPF06 U37
I 4Y 12 HB07 U36
I 4B 13 V07 U35
I 4A 14 MPF07 U34
C STR0BE 15 HBC1r U33
V VCC 16 VCC U32

```

```

*HB3
25LS157 16 2*Part name
C Sel 1 HBSel T43
I 1A 2 MPF00 T44
I 1B 3 V00 T45
I 1Y 4 HB00 T46
I 2A 5 MPF01 T47
I 2B 6 V01 T48
I 2Y 7 HB01 T49
G GND 8 GND T50
I 3Y 9 HB02 U50
I 3B 10 V02 U49
I 3A 10 MPF02 U48
I 4Y 12 HB03 U47
I 4B 13 V03 U46
I 4A 14 MPF03 U45
C STR0BE 15 HBC1r U44
V VCC 16 VCC U43

```

```

*U1
25LS09 16 2*Part name
C S 1 USel G21
@ Q0 2 U08 G22
I DOA 3 G08 G23
I DOB 4 T04 G24
I D1B 5 T05 G25
I D1A 6 G09 G26
@ Q1 7 U09 G27
G GND 8 GND G28
C CP 9 UCLK H28
@ Q2 10 U10 H27
I D2A 11 G10 H26
I D2B 12 T06 H25
I D3B 13 T07 H24
I D3A 14 G11 H23
@ Q3 15 U11 H22
V VCC 16 VCC H21

```

```

*U2
25LS09 16 2*Part name
C S 1 USel G32
@ Q0 2 U04 G33
I DOA 3 G04 G34
I DOB 4 T00 G35
I D1B 5 T01 G36
I D1A 6 G05 G37
@ Q1 7 U05 G38
G GND 8 GND G39
C CP 9 UCLK H39
@ Q2 10 U06 H38
I D2A 11 G06 H37
I D2B 12 T02 H36
I D3B 13 T03 H35
I D3A 14 G07 H34
@ Q3 15 U07 H33
V VCC 16 VCC H32

```

```

*U3
25LS09 16 2*Part name
C S 1 USel G43
@ Q0 2 U00 G44
I DOA 3 G00 G45
I DOB 4 U08 G46
I D1B 5 U09 G47
I D1A 6 G01 G48
@ Q1 7 U01 G49
G GND 8 GND G50
C CP 9 UCLK H50
@ Q2 10 U02 H49
I D2A 11 G02 H48
I D2B 12 U10 H47
I D3B 13 U11 H46
I D3A 14 G03 H45
@ Q3 15 U03 H44
V VCC 16 VCC H43

```

```

*V2
25LS09
C S
@ GO
I DOA
I DOB
I D1B
I D1A
@ Q1
@ Q2
C CP
@ Q2
I D2A
I D2B
I D3B
I D3A
@ Q3
V VCC

```

```

16 2*Part name
1 V21
2 V22
3 V23
4 V24
5 V25
6 V26
7 V27
8 V28
9 V29
10 V2A
11 V2B
12 V2C
13 V2D
14 V2E
15 V2F
16 V2G

```

```

*V2
25LS09
C S
@ GO
I DOA
I DOB
I D1B
I D1A
@ Q1
@ Q2
C CP
@ Q2
I D2A
I D2B
I D3B
I D3A
@ Q3
V VCC

```

```

16 2*Part name
1 V31
2 V32
3 V33
4 V34
5 V35
6 V36
7 V37
8 V38
9 V39
10 V3A
11 V3B
12 V3C
13 V3D
14 V3E
15 V3F
16 V3G

```

```

*V3
25LS09
C S
@ GO
I DOA
I DOB
I D1B
I D1A
@ Q1
@ Q2
C CP
@ Q2
I D2A
I D2B
I D3B
I D3A
@ Q3
V VCC

```

```

16 2*Part name
1 V41
2 V42
3 V43
4 V44
5 V45
6 V46
7 V47
8 V48
9 V49
10 V4A
11 V4B
12 V4C
13 V4D
14 V4E
15 V4F
16 V4G

```

```

*XM1
25LS157 16 2*Part name
C Sel 1 XMSEL C1
I 1A 2 UC8 C2
I 1B 3 V08 C3
@ 1Y 4 @108 C4
I 2A 5 UC9 C5
I 2B 6 V09 C6
@ 2Y 7 @109 C7
@ GND 8 GND C8
@ 3Y 9 @110 C8
I 3B 10 V10 D7
I 3A 11 U10 D6
@ 4Y 12 @111 D5
I 4B 13 V11 D4
I 4A 14 U11 D3
C STROBE 15 XMCIr D2
V VCC 16 VCC D1

```

```

*XM2
25LS157 16 2*Part name
C Sel 1 XMSEL E1
I 1A 2 U04 E2
I 1B 3 V04 E3
@ 1Y 4 @104 E4
I 2A 5 U05 E5
I 2B 6 V05 E6
@ 2Y 7 @105 E7
@ GND 8 GND F8
@ 3Y 9 @106 F8
I 3B 10 V06 F7
I 3A 11 U06 F6
@ 4Y 12 @107 F5
I 4B 13 V07 F4
I 4A 14 U07 F3
C STROBE 15 XMCIr F2
V VCC 16 VCC F1

```

```

*XM3
25LS157 16 2*Part name
C Sel 1 XMSEL G1
I 1A 2 U00 G2
I 1B 3 V00 G3
@ 1Y 4 @100 G4
I 2A 5 U01 G5
I 2B 6 V01 G6
@ 2Y 7 @101 G7
@ GND 8 GND G8
@ 3Y 9 @102 H8
I 3B 10 V02 H7
I 3A 11 U02 H6
@ 4Y 12 @103 H5
I 4B 13 V03 H4
I 4A 14 U03 H3
C STROBE 15 XMCIr H2
V VCC 16 VCC H1

```

```

*YH1
25LS157
C Sel
I 1A
I 1B
I 1Y
I 2A
I 2B
I 2Y
G GND
I 3Y
I 3B
I 3A
I 4Y
I 4B
I 4A
C STRB
V VCC

```

16	2*Part	name
1	YH1Sel	R1
2	U08	R2
3	V08	R3
4	#208	R4
5	U09	R5
6	V09	R6
7	#209	R7
8	GND	R8
9	#210	S8
10	V10	S7
11	U10	S6
12	#211	S5
13	V11	S4
14	U11	S3
15	YH1Clr	S2
16	VCC	S1

```

*YH2
25LS157
C Sel
I 1A
I 1B
I 1Y
I 2A
I 2B
I 2Y
G GND
I 3Y
I 3B
I 3A
I 4Y
I 4B
I 4A
C STRB
V VCC

```

16	2*Part	name
1	YH2Sel	T1
2	U04	T2
3	V04	T3
4	#204	T4
5	U05	T5
6	V05	T6
7	#205	T7
8	GND	T8
9	#206	U8
10	V06	U7
11	U06	U6
12	#207	U5
13	V07	U4
14	U07	U3
15	YH2Clr	U2
16	VCC	U1

```

*YH3
25LS157
C Sel
I 1A
I 1B
I 1Y
I 2A
I 2B
I 2Y
G GND
I 3Y
I 3B
I 3A
I 4Y
I 4B
I 4A
C STRB
V VCC

```

16	2*Part	name
1	YH3Sel	V1
2	U00	V2
3	V00	V3
4	#200	V4
5	U01	V5
6	V01	V6
7	#201	V7
8	GND	V8
9	#202	W8
10	V02	W7
11	U02	W6
12	#203	W5
13	V03	W4
14	U03	W3
15	YH3Clr	W2
16	VCC	W1

*DCD4
25L S07

Type	Contact	Width
C	1 GND	A1
I	2 GC2	A2
I	3 GCP2	A3
I	4 UO	A4
I	5 USel	A5
I	6 VO	A6
I	7 VSel	A7
C	8 GND	A8
C	9 CLK	B8
I	10	B7
I	11	B6
I	12	B5
I	13	B4
I	14	B3
I	15	B2
V	16 VCC	B1

*DCD5
25L S07

Type	Contact	Width
C	1 GND	R10
I	2 GBSel	R11
I	3 GBO	R12
I	4 GB1	R13
I	5 GBClr	R14
I	6 GAO	R15
I	7 GASel	R16
C	8 GND	R17
C	9 CLK	S17
I	10 GAClr	S16
I	11 GA1	S15
I	12 GCO	S14
I	13 GCP0	S13
I	14 GCP1	S12
I	15 GC1	S11
V	16 VCC	S10

*DCD6
25L S07

Type	Contact	Width
C	1 GND	T10
I	2 HBClr	T11
I	3 HB1	T12
I	4 HAO	T13
I	5 HASel	T14
I	6 HA1	T15
I	7 HAClr	T16
C	8 GND	T17
C	9 CLK	U17
I	10 HCO	U16
I	11 HCP0	U15
I	12 HC1	U14
I	13 HCP1	U13
I	14 HCP2	U12
I	15 HC2	U11
V	16 VCC	U10

*DCD7
25L S07

Type	Contact	Width
C	1 GND	V10
I	2 MPRND	V11
I	3 MO	V12
I	4 M1	V13
I	5 MXCLK	V14
I	6 M2	V15
I	7 MYCLK	V16
C	8 GND	V17
C	9 CLK	W17
I	10 MXSel	W16
I	11 M3	W15
I	12 MYSel	W14
I	13 M4	W13
I	14 HBO	W12
I	15 HBSel	W11
V	16 VCC	W10

*resistAPU

RESISTOR	2	2*Part name
V VCC	1	VCC G12
@ True	2	TRUE H12

CLK	34	88	S17	U17	W17
F00	012	E17			
F01	015	E18			
F02	H15	F19			
F03	H12	F18			
F04	C12	A17			
F05	C15	A18			
F06	D15	B19			
F07	D12	B18			
F208	B15	F16			
000	C48	045			
001	C49	048			
002	D50	H48			
003	D49	H45			
004	C37	034			
005	C38	037			
006	D39	H37			
007	D38	H34			
008	C26	023			
009	C27	026			
010	D28	H26			
011	59	D27	H23		
0108	F15	D25			
0208	D24	D36			
0308	D35	D47			
040	111	R15			

GA00	C43	A46			
GA01	C41	A49			
GA02	D42	B50			
GA03	D44	B47			
GA04	C32	A35			
GA05	C30	A38			
GA06	D31	B39			
GA07	D33	B36			
GA08	C21	A24			
GA09	C19	A27			
GA1	112	S15			
GA10	D20	B28			
GA11	A10 E12	A12 F13	B13 F11	B11 D22	E10 B25
GAC17	18	B22	B33	B44	S16
GASel	17	A21	A32	A43	R16
GB0	109	R12			
GB00	C44	E46			
GB01	C42	E49			
GB02	D43	F50			
GB03	D45	F47			
GB04	C33	E35			
GB05	C31	E38			
GB06	D32	F39			
GB07	D34	F36			
GB08	C22	E24			

GB09	C20	E27			
GB1	110	R13			
GB10	D21	F28			
GB11	D23	F25			
GBClr	23	F22	F33	F44	R14
GBSel	22	E21	E32	E43	R11
OC0	19 C45	A14 S14	E14	C23	C34
OC1	20 C46	A15 S11	E15	C24	C35
OC2	21 C47	A16 A2	E16	C25	C36
OCIN	60	D46			
OND	62 J44 LB E19 A39 R28 N39 G28 V50 T8 R17	63 J17 N8 C28 A50 R39 N50 G39 C8 V8 T10	J41 L17 C17 C39 E28 R50 T28 G50 E8 A1 T17	J42 N17 O17 C50 E39 S46 T39 V28 O8 A8 V10	J43 J8 A19 A28 E50 N28 T50 V39 R8 R10 V17
OBP0	113	S13			
OBP1	114	S12			
OBP2	115	A3			
H00	R48	V45			
H01	R49	V48			
H02	S50	W48			
H03	S49	W45			
H04	R37	V34			

H05	R38	V37			
H06	S39	W37			
H07	S38	W34			
H08	R26	V23			
H09	R27	V26			
H10	S28	W26			
H11	S27	W23			
H200	S24	S36			
H300	S35	S47			
HA0	104	T13			
HA00	R43	N46			
HA01	R41	N49			
HA02	S42	P50			
HA03	S44	P47			
HA04	R32	N35			
HA05	R30	N38			
HA06	S31	P39			
HA07	S33	P36			
HA08	R21	N24			
HA09	R19	N27			
HA1	105	T15			
HA10	S20	P28			
HA11	S22	P25			
HAC1r	50	P22	P33	P44	T16
HASel	49	N21	N32	N43	T14
H80	102	W12			

HB00	R44	T46			
HB01	R42	T49			
HB02	S43	U50			
HB03	S45	U47			
HB04	R33	T35			
HB05	R31	T38			
HB06	S32	U39			
HB07	S34	U36			
HB08	R22	T24			
HB09	R20	T27			
HB1	103	T12			
HB10	S21	U28			
HB11	S23	U25			
HB1r	55	U22	U33	U44	T11
HBSe1	54	T21	T32	T43	W11
HC0	51	R23	R34	R45	U16
HC1	52	R24	R35	R46	U14
HC2	53	R25	R36	R47	U11
HCRY	61	S25			
HCP0	106	U15			
HCP1	107	U13			
HCP2	108	U12			
I100	65	G13			
I101	66	G14			
I102	67	H14			
I103	68	H13			

I104	69	C13		
I105	70	C14		
I106	71	D14		
I107	72	D13		
I200	73	M11	N2	A45
I201	74	M14	N5	A48
I202	75	P15	P6	B49
I203	76	P12	P3	B46
I204	77	L11	L2	A34
I205	78	L14	L5	A37
I206	79	M15	M6	B38
I207	80	M12	M3	B35
I208	81	J11	J2	A23
I209	82	J14	J5	A26
I210	83	K15	K6	B27
I211	84	K12	K3	B24
I300	85	N12	N3	N45
I301	86	N15	N6	N48
I302	87	P16	P7	P49
I303	88	P13	P4	P46
I304	89	L12	L3	N34
I305	90	L15	L6	N37
I306	91	M16	M7	P38
I307	92	M13	M4	P35
I308	93	J12	J3	N23
I309	94	J15	J6	N26

T310	95	K16	K7	P27	
T311	96	K15	K4	P24	
M0	97	V12			
M1	98	V13			
M2	99	V15			
M3	100	W15			
M4	101	W13			
MPCLK	26	J45	J46		
MPF00	J47	E44	P23	T44	
MPF01	J48	A44	E47	T47	
MPF02	J49	A47	F48	U48	
MPF03	J50	B48	F45	U45	
MPF04	M50	B45	E33	T33	
MPF05	M49	A33	E36	T36	
MPF06	M48	A36	F37	U37	
MPF07	M47	B37	F34	U34	
MPF08	M46	B34	E22	T22	
MPF09	M45	A22	E25	T25	
MPF10	M44	A25	F26	U26	
MPF11	M43	B26	B23	F23	U23
MPLO0	J27	N44			
MPLO1	J28	N47			
MPLO2	J29	P48			
MPLO3	J30	P45			
MPLO4	J31	N33			
MPLO5	J32	N36			

MPL06	J33	P37			
MPL07	J34	P34			
MPL08	J35	N21			
MPL09	J36	N25			
MPL10	J37	P26			
MPLND	29	M25	V11		
M400	J26	N13			
M401	J25	N16			
M402	J24	P17			
M403	J23	P14			
M404	J22	L13			
M405	J21	L16			
M406	J20	M17			
M407	J19	M14			
M408	M19	J13			
M409	M20	J16			
MK10	M21	K17			
MK11	M22	K14			
MCLK	27	M23	V14		
MCLV	31	K11	M11	P11	
MCSel	30	J10	L10	N10	W16
MV00	M27	N4			
MV01	M28	N7			
MV02	M29	P8			
MV03	M30	P5			
MV04	M31	L4			

MY05	M32	L7			
MY06	M36	M8			
MY07	M37	M5			
MY08	M38	J4			
MY09	M39	J7			
MY10	M40	K8			
MY11	M41	K5			
MYCLX	28	M24	V16		
MYC1r	33	K2	M2	P2	
MYSel	32	J1	L1	N1	W14
NC	58	119			
0100	5	G4			
0101	6	G7			
0102	7	H8			
0103	8	H5			
0104	9	E4			
0105	10	E7			
0106	11	F8			
0107	12	F5			
0108	13	C4			
0109	14	C7			
0110	15	D8			
0111	16	D5			
0200	37	V4			
0201	38	V7			
0202	39	W8			

0203	40	W5			
0204	41	T4			
0205	42	T7			
0206	43	U8			
0207	44	U5			
0208	45	R4			
0209	46	R7			
0210	47	S8			
0211	48	S5			
0212	J38	M12	M26	B17	B16
	F17	D26	D37	D48	S26
	S37	S48			
T00	011	E13	G35		
T01	016	E11	G36		
T02	M16	F12	H36		
T03	M11	F14	H35		
T04	C11	A13	G24		
T05	C16	A11	G25		
T06	D16	B12	H25		
T07	D11	B14	H24		
TCLK	2	D17	H17		
TRUE	M12	J39	J40		
T901	1	C10	G10		
UD	116	A4			
UD0	E45	O44	V35	02	V2
UD1	E48	O49	V36	05	V5
UD2	F49	M49	W36	H6	W6

U03	F46	H44	W35	H3	W3
U04	E34	G33	V24	E2	T2
U05	E37	G38	V25	E3	T5
U06	F38	H38	W25	F6	U6
U07	F35	H33	W24	F3	U3
U08	E23	G22	G46	C2	R2
U09	E26	G27	G47	C5	R5
U10	F27	H27	H47	D6	S6
U11	F24	H22	H46	D3	S3
UCLK	24	H28	H39	H50	
US01	25	G21	G32	G43	A5
V0	117	A6			
V00	T45	V44	G3	V3	
V01	T48	V49	G6	V6	
V02	U49	W49	H7	W7	
V03	U46	W44	H4	W4	
V04	T34	V33	E3	T3	
V05	T37	V38	E6	T6	
V06	U38	W38	F7	U7	
V07	U35	W33	F4	U4	
V08	T23	V22	V46	C3	R3
V09	T26	V27	V47	C6	R6
V10	U27	W27	W47	D7	S7
V11	U24	W22	W46	D4	S4

VCC	G42	121	122	M35	M34
	M33	K10	M10	P10	K1
	M1	P1	D10	H10	B10
	F10	D19	D30	D41	B21
	B32	B43	F21	F32	F43
	S19	S30	S41	P21	P32
	P43	U21	U32	U43	H21
	H32	H43	W21	W32	W43
	D1	F1	H1	S1	U1
	W1	B1	S10	U10	W10
VCLK	56	W28	W39	W50	
VSe1	57	V21	V32	V43	A7
XPC17	4	D2	F2	H2	
XPG01	3	C1	E1	G1	
YPC17	36	S2	U2	W2	
YPS01	35	R1	T1	V1	
DONE					

APPENDIX B: MICRO PROGRAMS

Micro Programs

DOCUMENT		BUFFLD		OP DCBA		
RA	IA	T	M	J		
0	C	0	0	60	14 1406	LOADCD MACRO 60,L
2	60	0	2	61	6 0020	PA=20
4	61	7	0	62	14 7123	OUTPUT CLR
6	62	3	0	63	14 5001	WAIT 4,2
10	63	0	0	64	16 0365	IF OCT=3 SKIP
12	64	0	0	61	6 0010	INC PA GOTO A
14	65	0	0	3	6 0000	GOTO 3
16	66	0	0	60	6 0000	GOTO B

DOCUMENT		BUFFRD		OP DCBA		
RA	IA	T	M	J		
0	C	0	0	60	14 1407	LOADCD MACRO 60,10
2	60	0	2	61	6 0040	PA=10
4	61	7	0	62	14 7123	OUTPUT CLR
6	62	2	1	63	14 7120	INPUT
10	63	3	0	64	5 0513	MOVE DI=E1+I1+PD Y=3
12	64	0	0	65	16 0546	IF OCT=5 SKIP
14	65	0	0	61	6 0010	INC PA GOTO A
16	66	0	0	3	6 0000	GOTO 3
20	67	0	0	60	6 0000	GOTO B

DOCUMENT		CNTRL'D		OP DCBA		
RA	IA	T	M	J		
0	C	0	0	60	14 1406	LOADCD MACRO 60,L
2	60	0	2	61	6 0033	PA=33
4	61	7	0	62	14 7123	OUTPUT CLR
6	62	0	2	63	6 0002	PA=2
10	63	0	0	61	16 3402	IF SC(2)=0 GOTO A,OTW C
12	64	0	1	65	16 4500	SC(PA)=0
14	65	0	0	3	6 0000	GOTO 3
16	66	0	0	60	6 0000	GOTO B

DOCUMENT		COUNTLD		OP DCBA		
RA	IA	T	M	J		
0	C	0	0	60	14 1403	LOADCD MACRO 60,L
2	60	0	2	61	6 0031	PA=31
4	61	7	0	62	14 7123	OUTPUT CLR
6	62	0	0	3	6 0000	GOTO 3
10	63	0	0	60	6 0000	GOTO B

DOCUMENT	RA	TA	T	M	J	OP	DCBA	GLOBAL	INPTLD
0	C	0	0	60	14	1403	INPTLD		
2	60	0	2	61	6	0032	B		
4	61	7	0	62	14	7123			
6	62	0	0	3	6	0000			
10	63	0	0	60	6	0000	L		

LOADED MACRO 60,L
 PA=32
 OUTPUT CLR
 GOTO 3
 GOTO B

DOCUMENT	RA	TA	T	M	J	OP	DCBA	GLOBAL	RESMA
0	C	0	0	60	14	1403	RESMA		
2	60	0	2	61	6	0034	B		
4	61	7	0	62	14	7123			
6	62	0	0	3	6	0000			
10	63	0	0	60	6	0000	L		

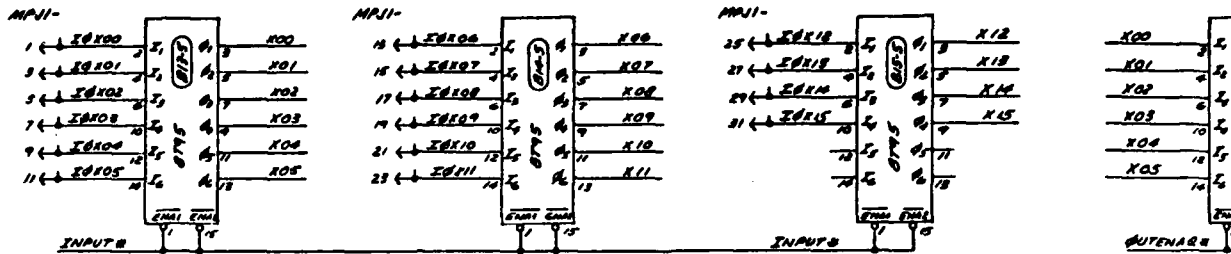
LOADED MACRO 60,L
 PA=34
 OUTPUT CLR
 GOTO 3
 GOTO B

DOCUMENT	RA	TA	T	M	J	OP	DCBA	GLOBAL	STATRD
0	C	0	0	60	14	1405	STATRD		
2	60	0	2	61	6	0051	B		
4	61	7	0	62	14	7123	A		
6	62	2	1	63	14	7120			
10	63	3	0	64	5	0513			
12	64	0	0	3	6	0000			
14	65	0	0	60	6	0000	L		

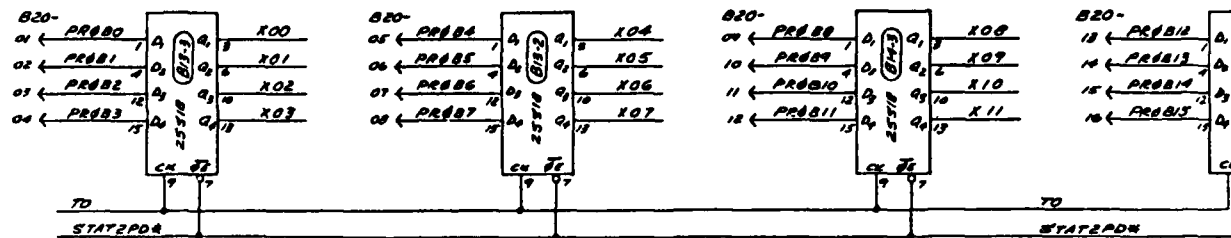
LOADED MACRO 60,L
 PA=51
 OUTPUT CLR
 INPUT
 MOVE DI+FI+TI+FI T=3
 GOTO 3
 GOTO B

APPENDIX C: MODULE ANALYZER LOGICAL DRAWINGS

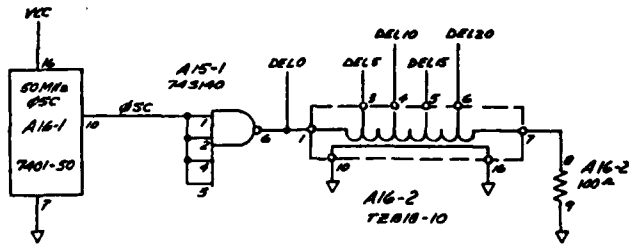
IOX BUSY



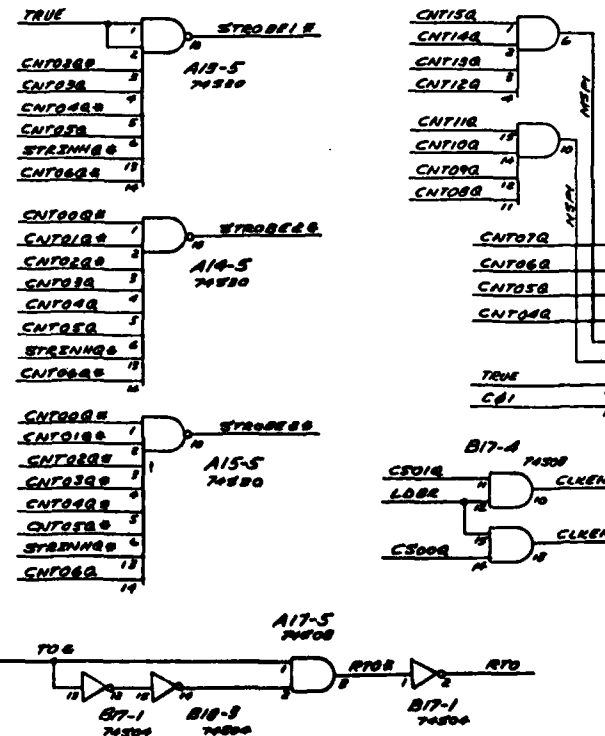
STATUS REGISTER

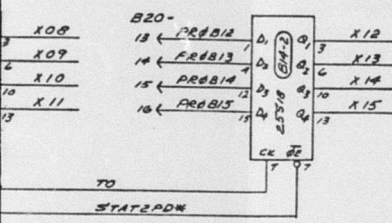
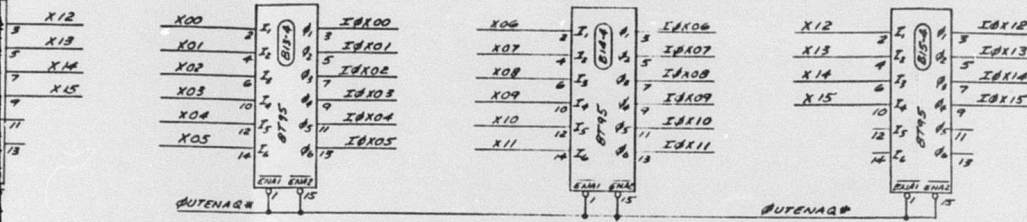


CLOCK

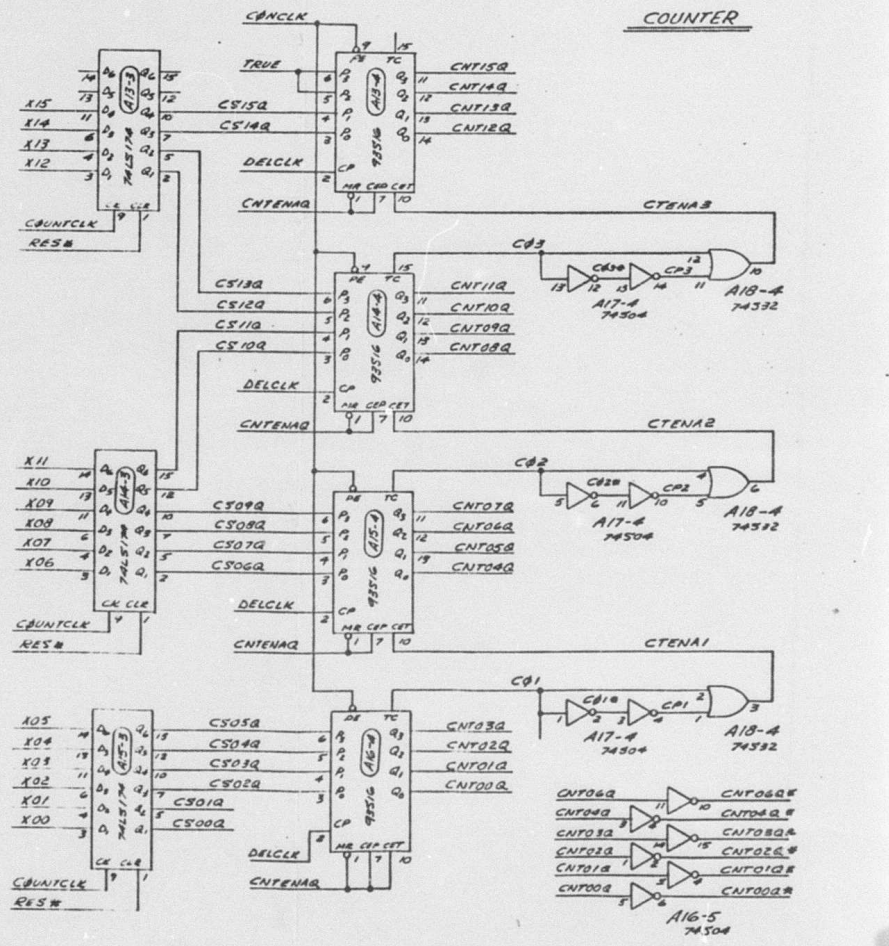
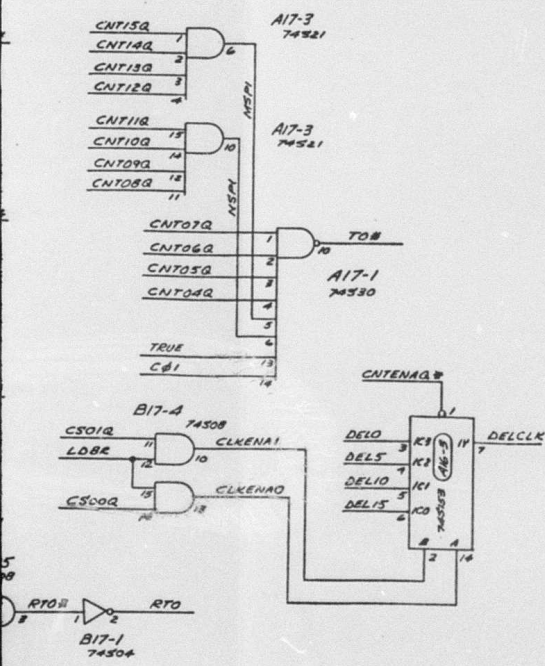


TIMING

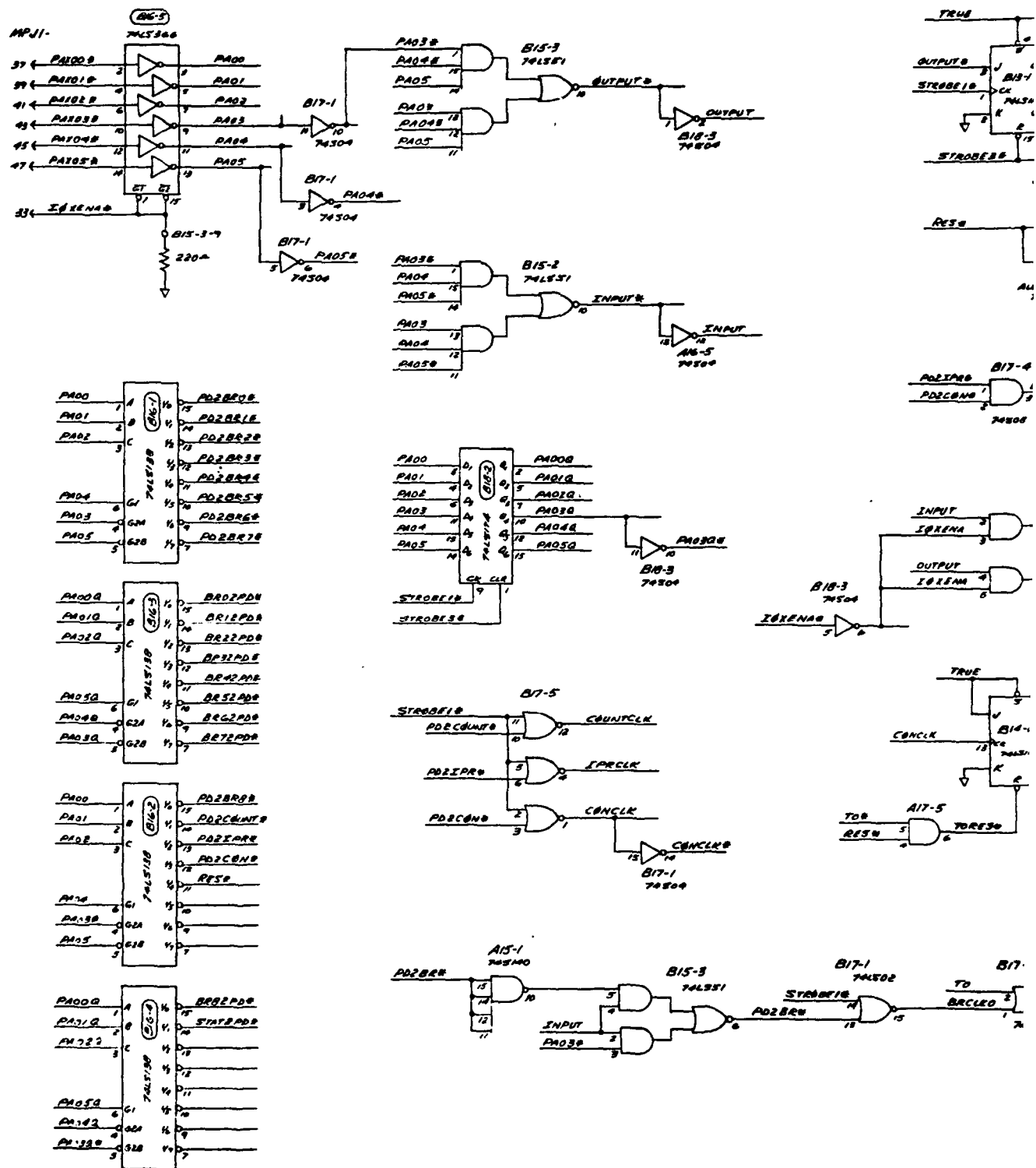




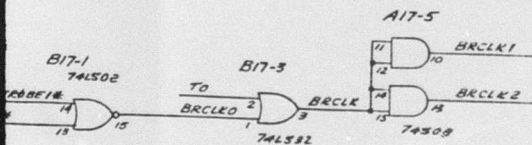
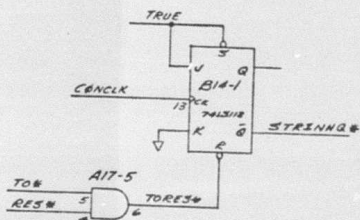
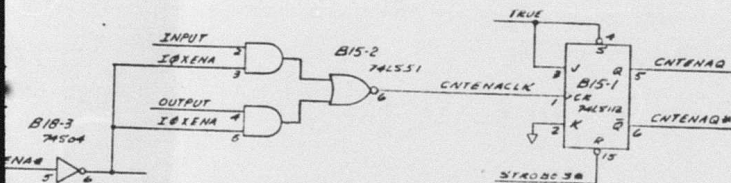
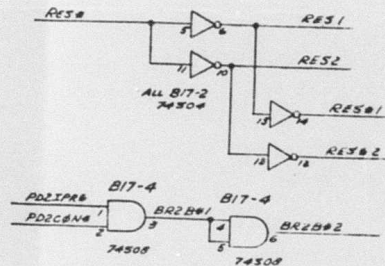
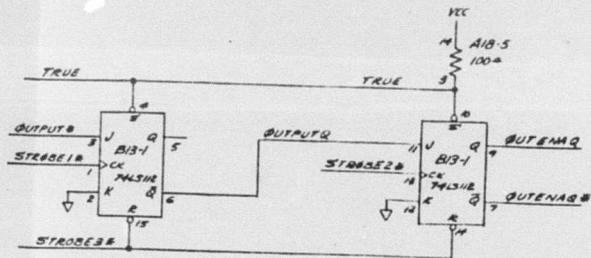
TIMING PULSES



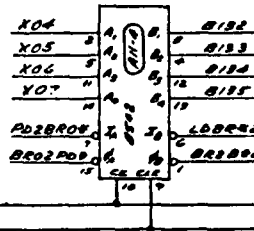
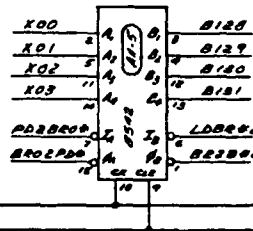
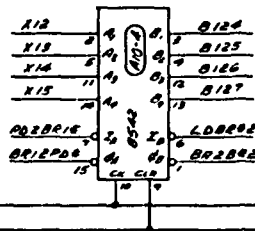
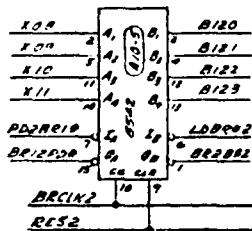
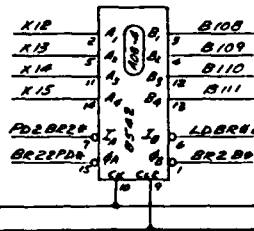
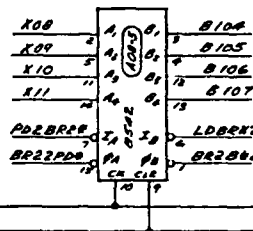
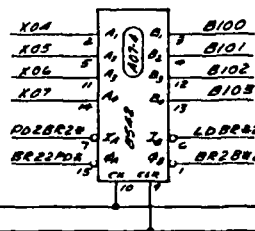
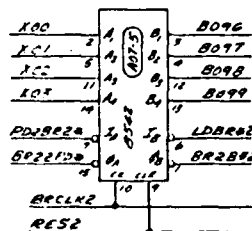
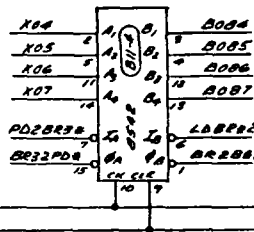
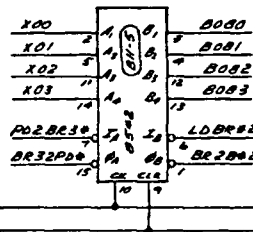
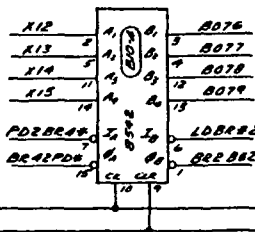
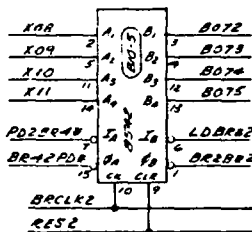
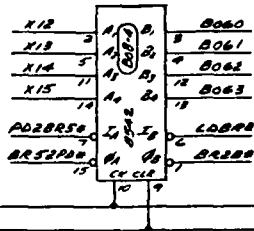
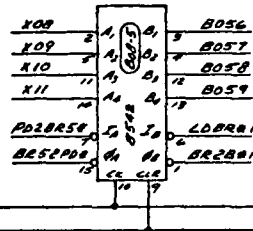
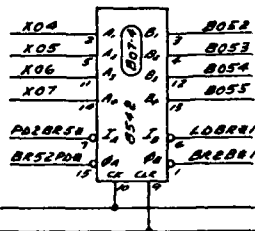
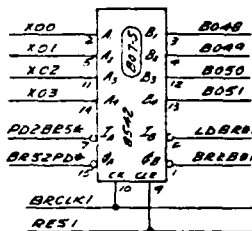
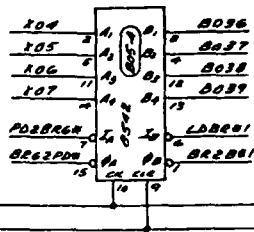
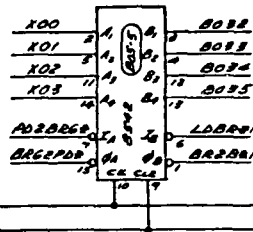
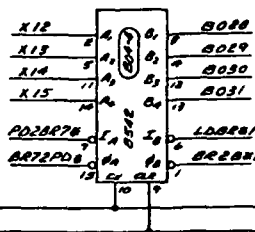
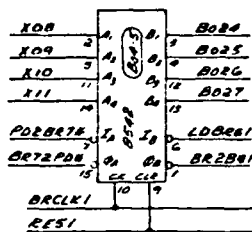
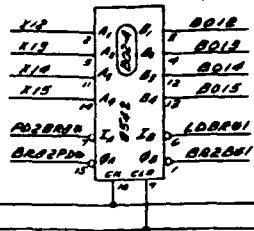
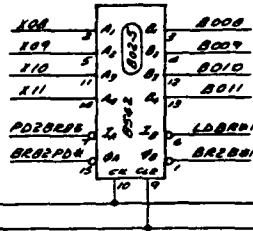
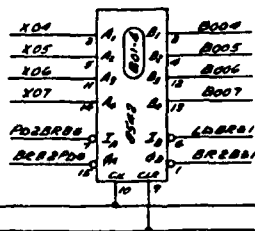
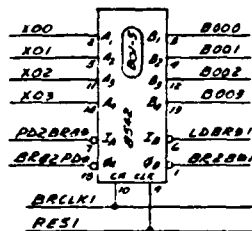
TOLERANCES (EXCEPT AS NOTED)			
DECIMAL	CH1	CULLER-HARRISON INC.	DATE
FRACTIONAL	MODULE ANALYZER	SEP 18	DRAWN BY
ANGULAR	TITLE	APPROVED BY	DATE
	TIMING LOGIC, X-BUS, STA REG		1/21/74
			DRAWING NUMBER
			MA02



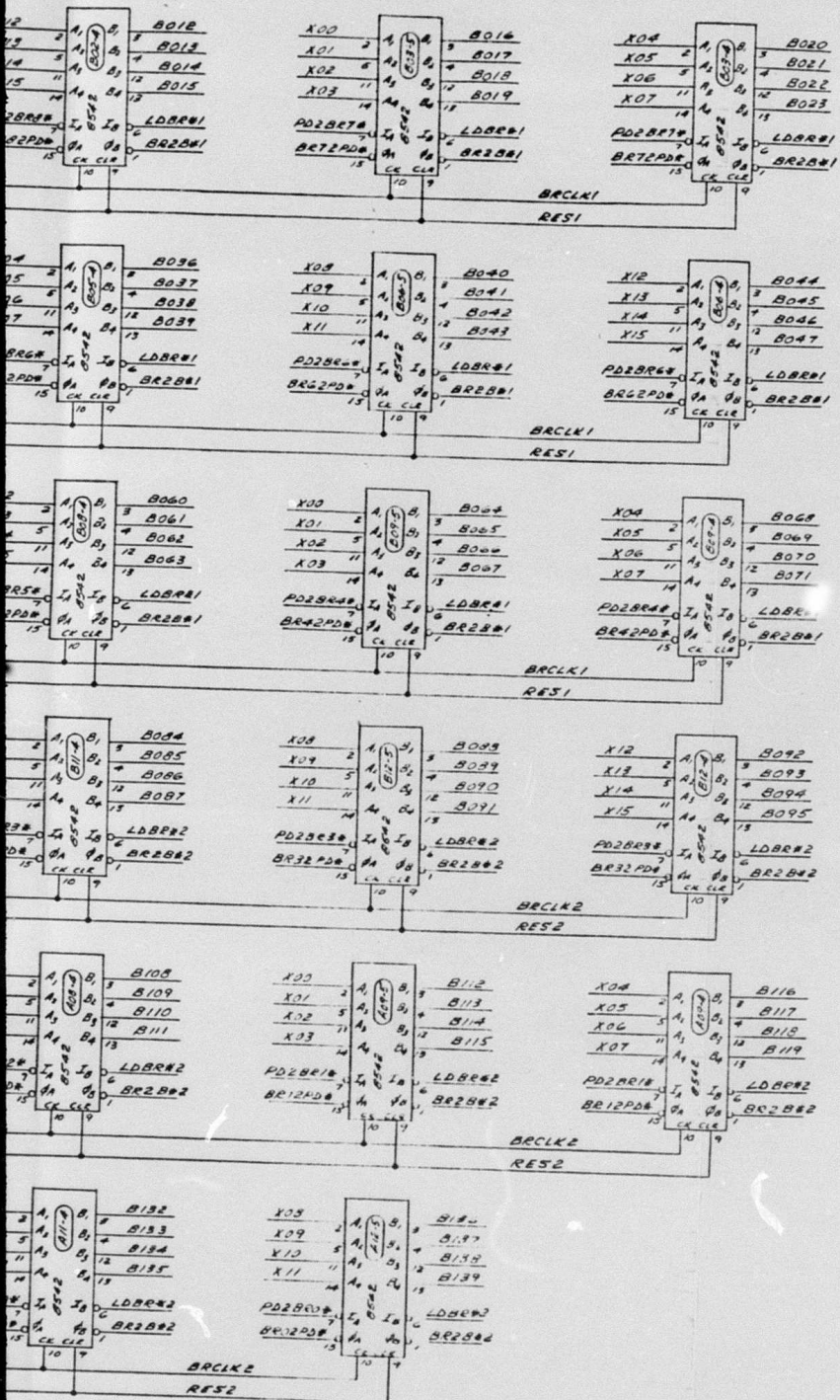
DATE	REV	BY	CHK



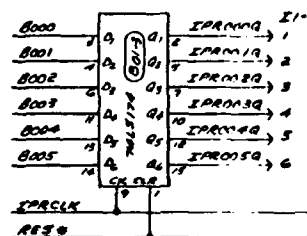
TOLERANCES (EXCEPT AS NOTED)	CHI CULLER-HARRISON INC.		
DECIMAL	MODULE ANALYZER	SCALE	DRAWN BY
FRACTIONAL	TITLE	DATE	APPROVED BY
ANGULAR	DATE	DRAWING NUMBER	
	7/26/79	MA03	



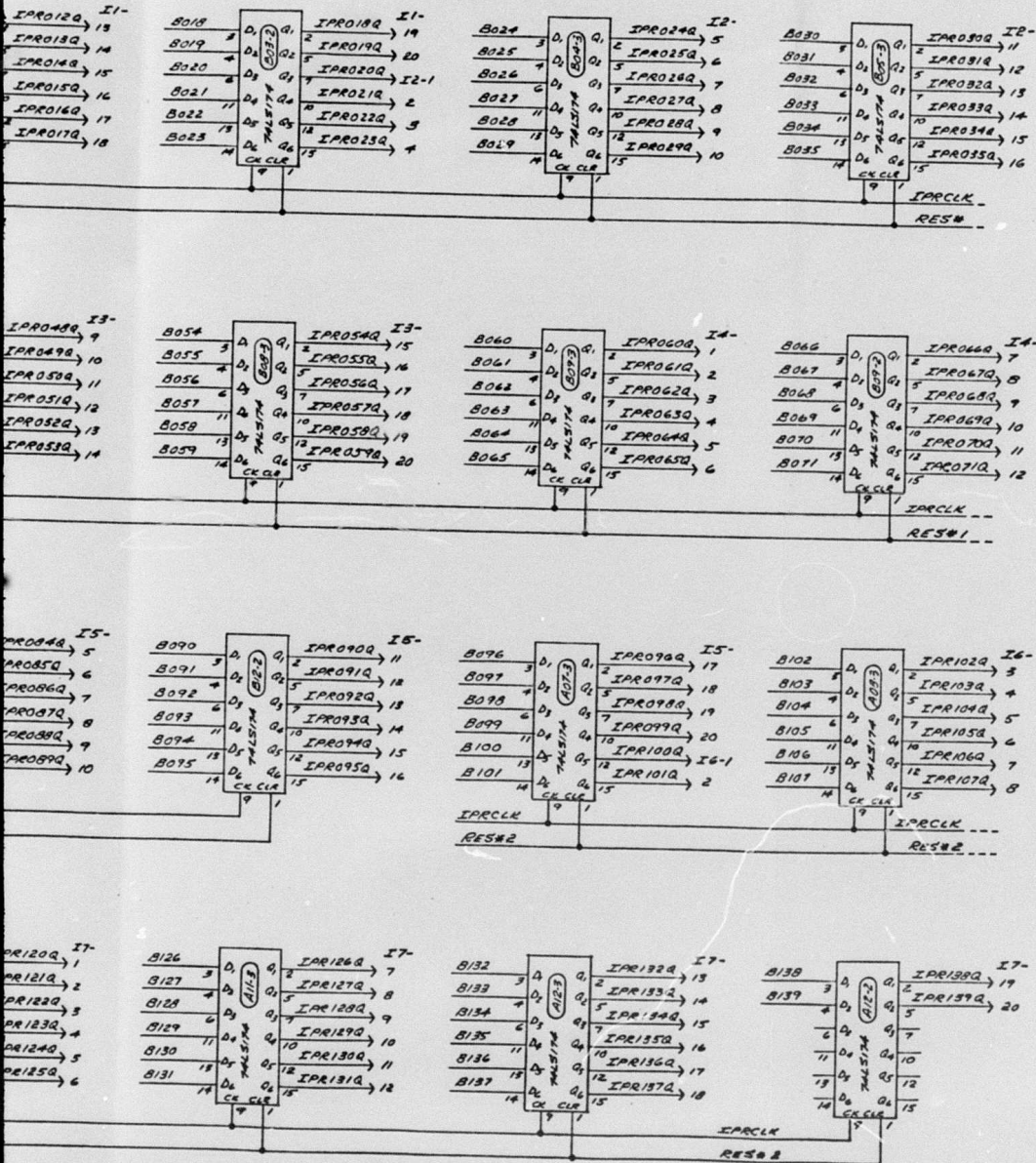
DATE	REVISION	REMARKS



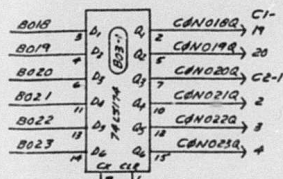
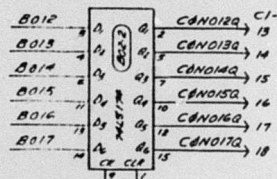
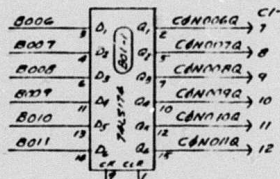
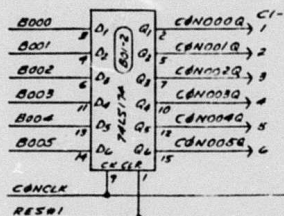
TOLERANCES (EXCEPT AS NOTED):		CHI CULLER-HARRISON INC.	
DECIMAL	MODULE ANALYZER	SCALE	DRAWN BY D. M. S. IMPROV
FRACTIONAL	TITLE	DATE	APPROVED BY
ANGULAR	7/24/78	DRAWING NUMBER	MA04



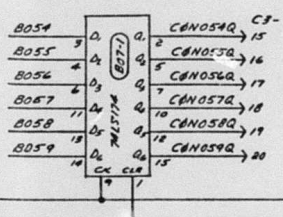
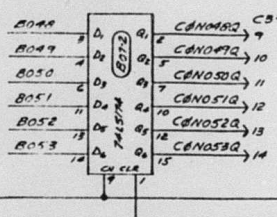
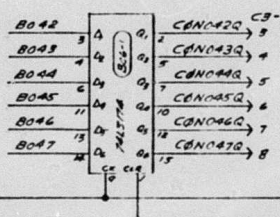
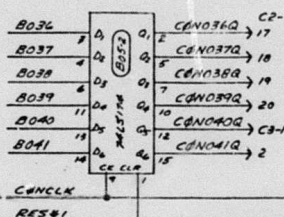
DATE	BY	REVISION RECORD	NO.	CL



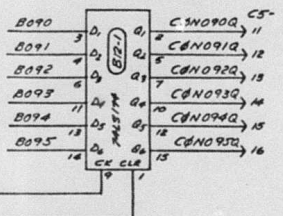
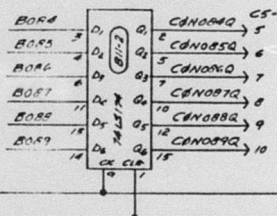
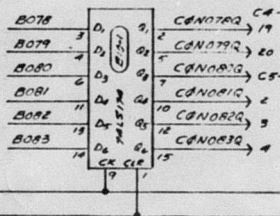
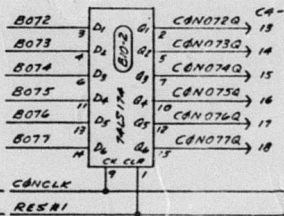
TOLERANCES (UNLESS OTHERWISE SPECIFIED)		CHI CULLER-HARRISON INC.	
FRACTIONAL	DECIMAL	SCALE	DRAWN BY C.H.I.
			APPROVED BY
FRACTIONAL	TITLE INPUT REGISTER		
ANALOG	DATE 1/24/78	DRAWING NUMBER	MA05



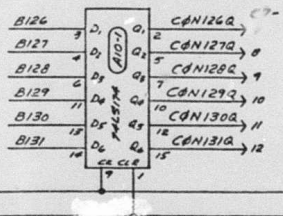
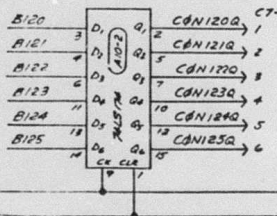
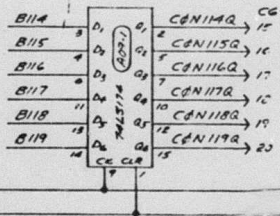
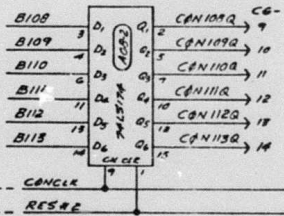
B024
 B025
 B026
 B027
 B028
 B029



B060
 B061
 B062
 B063
 B064
 B065

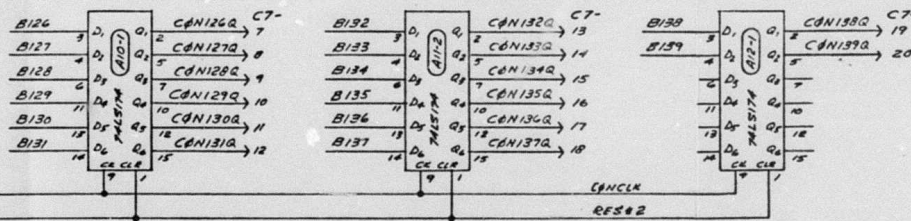
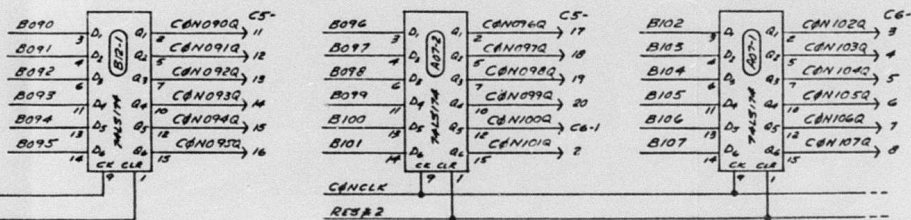
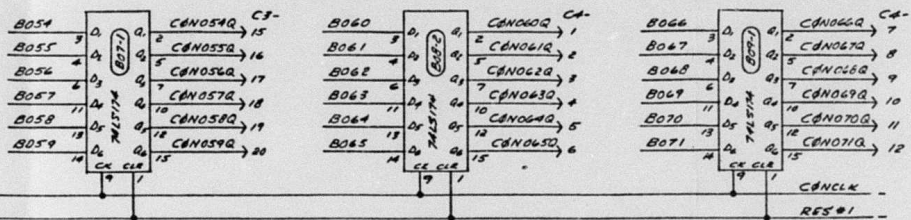
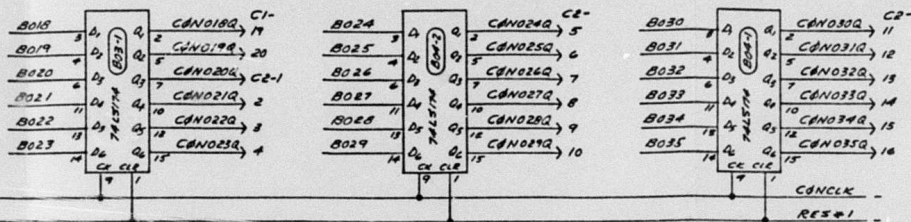


B096
 B097
 B098
 B099
 B100
 B101

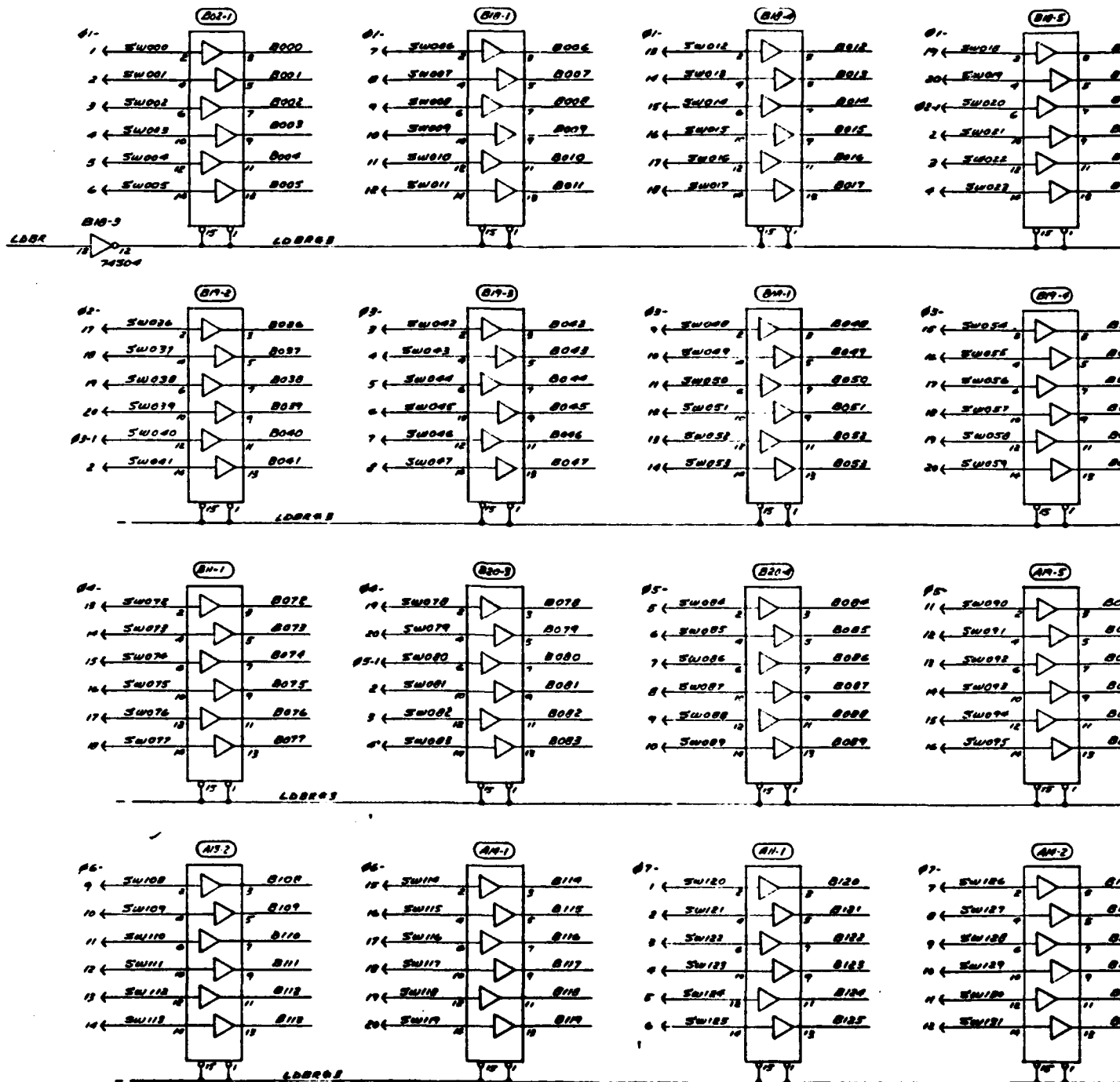


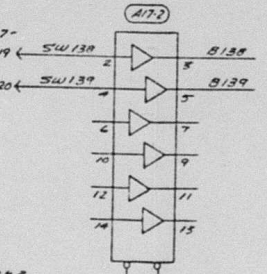
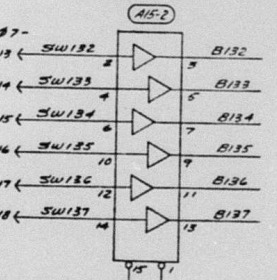
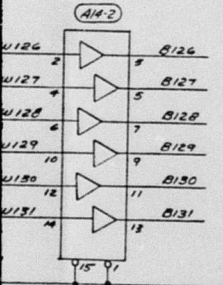
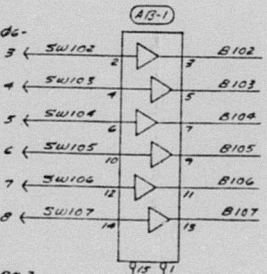
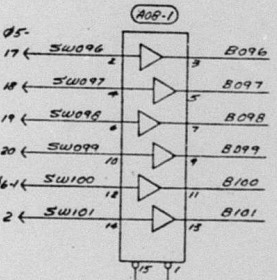
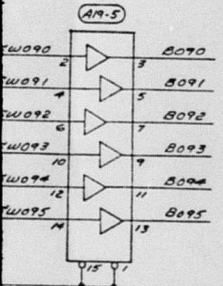
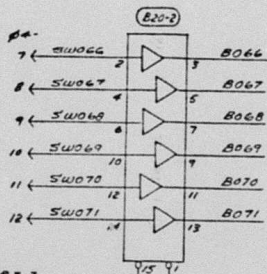
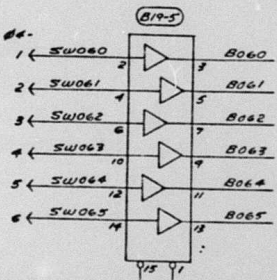
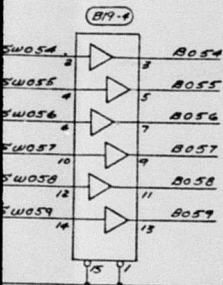
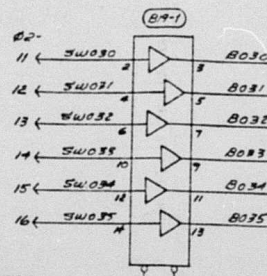
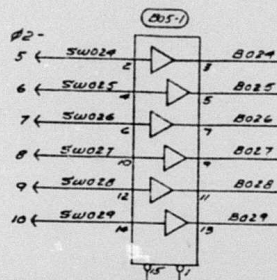
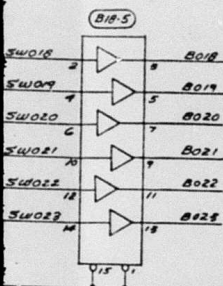
B132
 B133
 B134
 B135
 B136
 B137

DATE	BY	REVISION RECORD	DR	CH



TOLERANCES (EXCEPT AS NOTED)	CHI CULLER-HARRISON INC.		
DECIMAL 5	MODULE ANALYZER	SCALE -	DRAWN BY SUN SPANNAH
FRACTIONAL 5	TITLE CONTROL REGISTER		
ANGULAR 5	DATE 7/27/70	DRAWING NUMBER MA06	





TOLERANCES (EXCEPT AS NOTED)		CHI CULLER-HARRISON INC.	
DECIMAL	MODULE ANALYZER	SCALE	DRAWN BY
FRACTIONAL	TITLE	DATE	APPROVED BY
SWITCH BUFFERS		7/24/70	MAJ7

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CHI-QTR-301	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Structured Design for an LPC Array Processor Quarterly Technical Report Research on Speech Compression Prototype Development		5. TYPE OF REPORT & PERIOD COVERED Quarterly Technical Report May 1978 - August 1978
7. AUTHOR(s) Dr. Judith Bruckner Mr. Roger Russ Dr. Glen J. Culler Dr. Michael McCammon		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS Culler/Harrison, Inc. 150-A Aero Camino Goleta, California 93017		8. CONTRACT OR GRANT NUMBER(s) MDA903-78-C-0313
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Supply Service -- Washington Room 1D245, The Pentagon Washington, D.C. 20310		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Program Code: P8P10
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) DCASMA Oxnard 300 Esplanade Dr., Suite 990 Oxnard, California 93030		12. REPORT DATE September 1978
		13. NUMBER OF PAGES 46
		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce for sale to the general public. APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED. (A)		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES This research was supported by the Defense Advanced Research Projects Agency under ARPA Order No. 3625.		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) LPC Analysis Logical Modules LPC Synthesis Structured Design Array Processor Computer Architecture Logical Design		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The architecture of an array processor that is tailored to the requirements of LPC analysis and synthesis is presented. Interim results in the development of a structured design system are also given. <i>Keywords include:</i>		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)